1 Basic Lower Bounds on Approximability Via PCP and Gap Reductions

2 Introduction

In this chapter we will show that some problems are NP-hard to approximate. These are basic problems, like SAT, that will later be used to show other problems are hard to approximate. We can't use other hard-to-approximate problems in our reductions. The problems we discuss *are* those basic hard-to-approximate problems.

We use TSP as a running example. Recall that TSP is the following:

- Input: a weighted graph G and a number k.
- Determine if there is a Hamiltonian cycle of weight $\le k$.

For most of this book we have looked at *decision problems* where every instance has a *yes* or *no* answer. For example, TSP is a YES-NO question.

In the real world TSP is *not* a decision problem; indeed, in the real world one wants to *find* the optimal (minimum weight) cycle. This is the *function* version of TSP. We touched on this distinction in Chapter **??** and concluded (correctly) that, with regard to polynomial time, the decision problem and the function problem are equivalent. But let's get back to the real world. One way to cope with a problem being NP-hard is to approximate it. This concept only makes sense if we are talking about a *function*, not a *set*. In this chapter we will look at functions that are naturally associated to NP-complete problems and show that they are NP-hard to approximate. In this chapter we will show how to use *Gap Reductions* to get lower bounds on approximations contingent on P $\neq$ NP. We will then touch upon the Unique Games Conjecture, a point of great debate in modern times. There are lower bounds contingent on the Unique Games Conjecture (UGC) being true; however, the evidence for UGC is not as compelling as the evidence that P $\neq$ NP.

Def 2.1 An *optimization problem* consists of the following:

- The set of instances of the problem (e.g., the set of weighted graphs for TSP, the set of 3CNF formulas for MAX3SAT).

- For each instance: the set of possible solutions (e.g., the set of Hamiltonian cycles in that weighted graph, the set of truth assignments for the formula).
- For each solution: a nonnegative cost or benefit (e.g., the cost of the Hamiltonian cycle, the number of clauses that are satisfied).
- An objective: either min or max (e.g., min cost of a Hamiltonian cycle, max the number of clauses that are satisfied).

The goal of an optimization problem is to find a solution which achieves the objective: either minimize a cost or maximize a benefit.

Notation 2.2

1. Let A be a min-problem. Then $\mathrm{OPT}_A(x)$ is the cost of an optimal solution for instance x.
2. Let A be a max-problem. Then $\mathrm{OPT}_A(x)$ is the benefit of an optimal solution for the instance x.

Now we can define an NP-optimization problem. The class of all NP-optimization problems is called NPO; it is the optimization analog of NP.

Def 2.3 An NP-*optimization problem* is an optimization problem with the following additional requirements:

- All instances and solutions can be recognized in polynomial time (e.g., you can tell if a proposed cycle is Hamiltonian).
- All solutions are of length polynomial in the length of the instance which they solve (e.g., a Hamiltonian cycle is clearly of length polynomial in the size of the graph—it's actually shorter).
- The cost or benefit of a solution can be computed in polynomial time (given a Hamiltonian cycle in a weighted graph, one can easily find the weight of the cycle).

We can convert any NPO problem into an analogous decision problem in NP. For a min problem we ask *Is* $\mathrm{OPT}(x) \leq q$*?* and for a maximization problem we ask *Is* $\mathrm{OPT}(x) \geq q$*?* The optimal solution can serve as a short easily verified certificate of a "yes" answer, and so these analogous decision problems are in NP.

This means that NPO is, in some sense, a generalization of NP problems.

Convention 2.4 For the rest of this chapter *problem* means NPO *problem*. When A and B are mentioned they are NPO problems. We will often say whether A is a min-problem or a max-problem.

Note that when we speak of solving an NPO problem we only mean finding OPT(x) which is the cost or benefit of the optimal solution. So we are not quite in the real world: for us a solution to the TSP problem is just to say how much the min Ham cycle costs, not to find it. However, this will suit our purposes:

- We will show that it's hard to approximate OPT up to certain factors. Hence clearly it will be hard to find an approximate solution.
- All of the algorithms in the literature for approximation problems actually do find an approximate solution.

3 Approximation Algorithms

Let's say you are trying to solve TSP. Let TSP be the function that returns the cost of the optimal Hamiltonian cycle. Lets say you have an algorithm that will, on input G, output a Hamiltonian cycle of weight $\le 2\mathrm{TSP}(G)$. So it's at worst twice-the-optimal. Is that good? Can you prove that no algorithm is better unless P = NP? Before asking these questions we need to define our terms.

Def 3.1 In the two definitions below, ALG is a polynomial time algorithm and $c \ge 1$ is a constant. (We will later generalize to the case where c is a function.)

- Let A be a min-problem. ALG is a *c-approximation algorithm for A* if, for all valid instances x,

$$\mathrm{ALG}(x) \le c\mathrm{OPT}(x).$$

- Let A be a max-problem. ALG is a *c-approximation algorithm for A* if, for all valid instances x,

$$\mathrm{OPT}(x) \le c\mathrm{ALG}(x).$$

(Note that if there is an algorithm that, on input a 3CNF formula outputs a number that is $\le \frac{7}{8}$MAX3SAT(ϕ), then this is called an $\frac{8}{7}$-approximation. We personally do not like this notation and will avoid using it.)

In some sense an approximation algorithm is doing pretty well if it is a c-approximation algorithm with some constant value c. But sometimes, we can do even better! There are cases where, for all $\epsilon > 0$, there is a $(1+\epsilon)$-approximation.

Def 3.2

1. Let A be a min-problem. A *polynomial time approximation scheme (*PTAS*)* for A is an algorithm that takes as input (x, ϵ) (x an instance of A and $\epsilon > 0$) and outputs a solution that is $\le (1+\epsilon)\mathrm{OPT}(x)$. The only runtime constraint is that if we fix ϵ, the PTAS must run in time polynomial in the size of the remaining input. Note that this allows very bad runtimes in terms of ϵ. For example, a PTAS can run in time n^{1/ϵ^2} because for any given value of ϵ this is a polynomial runtime.

2. Let A be a max-problem. A *polynomial time approximation scheme* (PTAS) for A is an algorithm that takes as input (x, ϵ) (x an instance of A and $\epsilon > 0$) and outputs a solution that is $\ge (1-\epsilon)\mathrm{OPT}(x)$. We have the same runtime constraint as in part 1.

We can now define several complexity classes:

Def 3.3

1. The class PTAS is the set of all problems for which a PTAS exists. We use the term PTAS for both the type of an approximation algorithm and the set of all problems that have that type of approximation algorithm.

2. Let A be a min-problem. $A \in$ APX if there is a constant $c \ge 1$ and an algorithm M such that $M(x)$ is $\le c \times \mathrm{OPT}(x)$. (This is just a c-approximation; however, we phrase it this way so that you will see the other classes are variants of it.)

3. Let A be a max-problem. $A \in$ APX if there is a constant $c \ge 1$ and an algorithm M such that $M(x)$ is $\ge \frac{1}{c} \times \mathrm{OPT}(x)$.

4. Let A be a min-problem. $A \in \text{LOG-APX}$ if there is a constant $c \geq 1$ and an algorithm M such that $M(x)$ is $\leq c \times \log x \times \text{OPT}(x)$.

5. Let A be a max-problem. $A \in \text{LOG-APX}$ if there is a constant c and an algorithm M such that $M(x)$ is $\geq \frac{1}{c \log x} \times \text{OPT}(x)$.

6. Let A be a min-problem. $A \in \text{POLY-APX}$ if there is a polynomial p and an algorithm M such that $M(x)$ is $\leq p(x) \times \text{OPT}(x)$.

7. Let A be a max-problem. $A \in \text{POLY-APX}$ if there is a polynomial p and an algorithm M such that $M(x)$ is $\geq \frac{1}{p(x)} \times \text{OPT}(x)$.

The following examples are known.

Example 3.4 All of our examples are variants of TSP.

1. The *metric* TSP problem is the TSP problem restricted to weighted graphs that are symmetric and satisfy the triangle inequality: $w(x, y) + w(y, z) \geq w(x, z)$. There is an algorithm discovered independently by Christofides [7] (in 1976) and Serdyukov [30] (in 1978) that gives a $\frac{3}{2}$-approximation to the metric TSP problem. Hence the metric TSP problem is in APX.

2. Karlan, Klein, Oveis-Gharan [17], in 2020, obtained the first improvement over the $\frac{3}{2}$-approx. They showed that there is a $(\frac{3}{2} - \epsilon)$-approximation to the metric TSP problem where $\epsilon > 10^{-36}$. This does not improve the class that metric TSP is in—it is still in APX— but it is interesting that one can do better than $\frac{3}{2}$ which was, until this result, a plausible limit on approximation.

3. The *Euclidean* TSP problem is the TSP problem when the graph is a set of points in the plane and the weights are the Euclidean distances. Arora [3] and Mitchell [27], in 1998, independently showed a PTAS for the Euclidean TSP problem. Both of their algorithms will, on input n points in the plane (which defines the weighted graph) and ϵ, produce a $(1 + \epsilon)$-approximation in time $O(n(\log n)^{O(1/\epsilon)})$.

4. Arora and Mitchell actually have an algorithm that works on n points in R^d that runs in time $O(n(\log n)^{O(\sqrt{d}/\epsilon)^{d-1}})$.

4 The Basic Hard-to-Approximate Problem

The following are basic hard-to-approximate problems. We include both the upper and the lower bounds. We will show the lower bounds, or weaker versions of them, by using the PCP characterization of NP (to be discussed later). All of the lower bounds are under the assumption P $\neq$ NP.

1. TSP $\notin$ POLY-APX.
2. CLIQ $\in$ POLY-APX $-$ LOG-APX.
3. SETCOVER $\in$ LOG-APX $-$ APX.
4. MAX3SAT $\in$ APX $-$ PTAS.

From the results enumerated above we have the following.

Theorem 4.1 *If* P $\neq$ NP *then*

$$\text{PTAS} \subset \text{APX} \subset \text{LOG-APX} \subset \text{POLY-APX}.$$

We will discuss all four of the results in the order given above. There will several sections between the TSP lower bound and the CLIQ lower bound since CLIQ uses the PCP machinery. MAX3SAT also uses it. SETCOVER uses a different machinery and will be discussed only briefly.

5 Lower Bounds on Approximating TSP

Recall that, by Example 3.4, the metric TSP problem (where $w(a,b) + w(b,c) \leq w(a,c)$) is in APX. What about TSP problems without that condition? We show that if TSP $\in$ POLY-APX, then P = NP. Informally, we will map instances G of HAM CYCLE to instances G' of TSP such that

If $G \in$ HAM CYCLE then G' has a very cheap Hamiltonian Cycle.

If $G \notin$ HAM CYCLE then G''s cheapest Hamiltonian Cycle is quite costly.

We will then use the alleged approximation algorithm for TSP to determine which is the case. This is called a *Gap Reduction* because of the large gap between the optimal routes.

Theorem 5.1 *If* TSP $\in$ POLY-APX *then* P = NP.

Proof:

Assume, by way of contradiction, that TSP $\in$ POLY-APX with polynomial $p(n)$. To avoid notational clutter we call this *the approx alg.* We use this assumption to show that HAM CYCLE $\in$ P.

Let $c(n)$ be a polynomial to be named later. We give a reduction that (1) maps Hamiltonian graphs to instances G' of TSP with $\text{TSP}(G') = n$, and (2) maps non-Hamiltonian graphs to instances G' of TSP with $\text{TSP}(G') \geq c(n)$.

Here is an algorithm for HAM CYCLE.

1. Input $G = (V, E)$, an unweighted graph.
2. Create an instance G' of TSP as follows: (1) if $e \notin E$ then give e weight $c(n)$, (2) if $e \in E$ then give e weight 1.
3. (This is a comment, not part of the algorithm.)
   (a) If $G \in$ HAM CYCLE then $\text{TSP}(G') \leq n$ since you can just use the Hamiltonian cycle.
   (b) If $G \notin$ HAM CYCLE then $\text{TSP}(G') \geq c(n)$ since any cycle in G' will have to use at least one edge of cost $c(n)$ (actually $\text{TSP}(G') \geq c(n) + n - 1$ but this is not needed).
4. Run the approx algorithm on G'.
5. (This is a comment, not part of the algorithm.)
   (a) If $G \in$ HAM CYCLE then the approx alg run on G' returns a route of size $\leq np(n)$.
   (b) If $G \notin$ HAM CYCLE then the approx alg run on G' returns a route of size $\geq c(n)$.

   To ensure these cases do not overlap we pick $c(n) > np(n)$.
6. If the approx alg outputs a number $\leq np(n)$ then output YES. If the approx alg outputs a number $> c(n)$ then output NO. By the commentary in the algorithm, no other case will occur.

▮

The proof of Theorem 5.1 took G and *produced a gap.* We actually showed the following:

Theorem 5.2 *Let $c(n) = np(n)$ where $p(n)$ is a polynomial of degree ≥ 1. Let $c(n)$-GAP-*TSP *be defined as follows. Given a* TSP *problem $G = (V, E, w)$ where you are promised that exactly one of the following occurs:*

- *There is a solution of cost n.*
- *All solutions have cost $\geq c(n)$.*

Then, for all $c(n)$, HAM CYCLE *reduces to $c(n)$*-GAP-TSP.

6 The Gap Lemmas

In this section we prove two easy lemmas that show how to use a reduction that causes a gap (like the one in Theorem 5.1) to obtain a lower bound on approximation algorithms. This first lemma is for max-problems, and the second one is for min-problems. The proofs are similar, hence the proof of the second one is omitted.

Convention 6.1 We will often use the notation $|y|$. This is the size of y; however, we will use *size* in a different way for different inputs.

1. If y is a string then $|y|$ is the length of y.
2. If y is a graph then y is the number of vertices.
3. If y is a 3CNF formula then y might be either the number of variables or the number of clauses depending on our application.

Def 6.2 Let g be a max-problem (e.g., CLIQ). Let $a(n)$ and $b(n)$ be functions from N to N such that $\frac{b(n)}{a(n)} < 1$. Then GAP$(g, a(n), b(n))$ is the following problem.

Problem 6.3
INSTANCE: y for which you are promised that either $g(y) \geq a(|y|)$ or $g(y) \leq b(|y|)$.
QUESTION: Determine which is the case.

Lemma 6.4 *Let A be an* NP*-hard set. Let g be a max-problem. Let $a(n)$ and $b(n)$ be functions from* N *to* N *such that (1) $\frac{b(n)}{a(n)} < 1$, and (2) b is computable in time polynomial in n. Assume there exists a polynomial time reduction that maps x to y such that the following occurs:*

- *If $x \in A$ then $g(y) \geq a(|y|)$.*
- *If $x \notin A$ then $g(y) \leq b(|y|)$.*

Then:

1. $\mathrm{GAP}(g, a(n), b(n))$ *is* NP*-hard (this follows from the premise).*
2. *If there is an approximation algorithm for g that, on input y, returns a number $> \frac{b(|y|)}{a(|y|)} g(y)$, then* $\mathrm{P} = \mathrm{NP}$.

Proof: We just prove part 2.

We use the reduction and the approximation algorithm to obtain $A \in \mathrm{P}$. Since A is NP-hard we obtain $\mathrm{P} = \mathrm{NP}$.

Algorithm for A

1. Input x.
2. Run the reduction on x to get y.
3. Run the approximation algorithm on y.
4. (This is a comment and not part of the algorithm.)

   $x \in A \rightarrow g(y) \geq a(|y|) \rightarrow$ approx on y returns $> \frac{b(|y|)}{a(|y|)} a(|y|) = b(|y|)$.

   $x \notin A \rightarrow g(y) \leq b(|y|) \rightarrow$ approx on y returns $\leq b(|y|)$.
5. If the approx returns a number $> b(|y|)$ then output YES. Otherwise output NO. (This is the step where we need $b(|y|)$ to be computable in time polynomial in $|y|$.)

▮

We now look at min-problems.

Def 6.5 Let g be a min-problem (e.g., TSP). Let $a(n)$ and $b(n)$ be functions from N to N such that $\frac{b(n)}{a(n)} > 1$. Then $\mathrm{GAP}(g, a(n), b(n))$ is the following problem.

Problem 6.6

INSTANCE: y for which you are promised that either $g(y) \le a(|y|)$ or $g(y) \ge b(|y|)$.

QUESTION: Determine which is the case.

We now state a lemma that is useful for obtaining lower bounds on approximation min-problems. The proof is similar to that of Lemma 6.4 and hence is omitted.

Lemma 6.7 *Let A be an* NP*-complete set. Let g be a min-problem. Let $a(n)$ and $b(n)$ be functions from N to N such that (1) $\frac{b(n)}{a(n)} > 1$, and (2) b is computable in time polynomial in n. Assume there exists a polynomial time reduction that maps x to y such that the following occurs:*

- *If $x \in A$ then $g(y) \le a(|y|)$.*
- *If $x \notin A$ then $g(y) \ge b(|y|)$.*

Then:

1. $\mathrm{GAP}(g, a(n), b(n))$ *is* NP*-hard (this follows from the premise).*
2. *If there is an approximation algorithm for g that, on input y, returns a number $< \frac{b(|y|)}{a(|y|)} g(y)$, then* $\mathrm{P} = \mathrm{NP}$.

Def 6.8 We will refer to reductions like the ones in Lemma 6.4 and 6.7 as *Gap Reductions with ratio* $\frac{b(n)}{a(n)}$.

7 The PCP Machinery

In this section we discuss a characterization of NP in terms of *Probabilistically Checkable Proofs*. This characterization has a rather long proof that we will omit. However, once we have the characterization we will use it to construct gap reductions which will show some approximations are NP-hard.

Recall the following notation and definition.

Notation 7.1 Let $\exists^p y$ mean there exists y such that $|y|$ is of length polynomial in $|x|$, where x is understood. Let $\forall^p y$ mean for all y such that $|y|$ is of length polynomial in $|x|$, where x is understood.

Def 7.2 $A \in \text{NP}$ if there exists a polynomial predicate B such that

$$A = \{x : (\exists^p y)[B(x,y)]\}.$$

We want to rewrite this and modify it.

Def 7.3

1. An *Oracle Turing Machine–bit access* (henceforth OTM-BA) is an Oracle Turing Machine where (1) the oracle is a string of bits, and (2) the requests for the bits is made by writing down the address of the bit. By convention, if the string is s long and a query is made for bit $t > s$ then the answer is NO.

2. We denote an oracle Turing machine by $M^{()}$. If $M^{()}$ is an Oracle Turing Machine and y is the string being used for the oracle, and x is an input, we denote the computation of $M^{()}$ on x with oracle y by $M^y(x)$.

3. A *Polynomial OTM-BA (POTM-BA)* is an OTM-BA that runs in polynomial time. Note that a POTM-BA can use an oracle string of length 2^{poly} since it can write down that it wants bit position (say) 2^{n^2} with n^2 bits.

4. We give two equivalent definitions of a *Randomized POTM-BA (RPOTM-BA)*. One is intuitive and the other is better for proofs.

   (a) A *Randomized POTM-BA (RPOTM-BA)* is a POTM-BA that is allowed to flip coins. So there will be times where, rather than do STEP A it will do STEP A with probability (say) 1/3 and STEP B with probability 2/3. Hence we cannot say *The machine accepts x using oracle bit string y* but we can say *The machine will accept x using oracle bit string y with probability* ≥ 0.65.

   (b) Note that for a RPOTM-BA computation many coins are flipped and are used. We can instead think of the string of coin flips as being part of the input, and then asking what fraction of the inputs accept. Formally, a *Randomized POTM-BA (RPOTM-BA)* is a POTM-BA that has 2 inputs x, τ. We will be concerned with the fraction of τ's ($|\tau|$ will be a function of $|x|$) for which $M^y(x,\tau)$ accepts. We will refer to this as *the probability that x with oracle* *bit string y is accepted* since we think in terms of the string τ being chosen at random. We will refer to τ as a string of coin flips.

The following is an alternative definition of NP.

Def 7.4 $A \in \mathrm{NP}$ if there exists a POTM-BA $M^{()}$ such that:
$x \in A \rightarrow (\exists^p y)[M^y(x) = 1]$
$x \notin A \rightarrow (\forall^p y)[M^y(x) \neq 1]$

If $x \in A$ then we think of y as being the EVIDENCE that $x \in A$. This evidence is short (only $p(|x|)$ long) and checkable in poly time. Note that the computation of $M^y(x)$ may certainly use all of the bits of y. What if we (1) restrict the number of bits of the oracle that the computation can look at, and (2) use an RPOTM-BA?

Def 7.5 Let $q(n)$ and $r(n)$ be monotone increasing functions from N to N. A *$r(n)$-random $q(n)$-query RPOTM-BA* $M^{()}$ is a RPOTM-BA where, for all y and for all x of length n, $M^y(x)$ flips $r(n)$ coins and makes $q(n)$ queries.

Def 7.6 Let $r(n)$ and $q(n)$ be monotone increasing functions from N to N and $\epsilon(n)$ be a monotone decreasing function from N to $[0,1]$. $A \in \mathrm{PCP}(r(n), q(n), \epsilon(n))$ if there exists an $r(n)$-random, $q(n)$-query RPOTM-BA $M^{()}$ such that, for all n, for all $x \in \{0,1\}^n$, the following holds.

1. If $x \in A$ then there exists y such that, for all τ with $|\tau| = r(n)$, $M^y(x,\tau)$ accepts. In other words, the probability of acceptance is 1.
2. If $x \notin A$ then for all y at most $\epsilon(n)$ of the τ's with $|\tau| = r(n)$ make $M^y(x,\tau)$ accept. In other words, the probability of acceptance is $\leq \epsilon(n)$.
3. (This item is not formally needed.) One of the two cases above must happen. That is, there will never be a case where (say) $\epsilon(n) < \frac{1}{2}$ and the probability of acceptance is $2\epsilon(n)$.

We are only going to be concerned with $r(n) = O(\log n)$ and $q(n) = O(1)$ or $O(\log n)$. We will see below that we can assume $|y| = 2^{q(n)+r(n)}$, which is poly in n.

Note 7.7 The queries are made adaptively. This means that the second question asked might depend on the answer to the first. Hence if $M^y(x, \tau)$ asks $O(q(n))$ questions then the total number of questions possible to ask is $2^{O(q(n))} - 1$. Since there are $2^{O(r(n))}$ values of z there are a total of roughly $2^{O(q(n)+r(n))}$ queries that can be asked. Hence we can take $|y| = 2^{q(n)+r(n)}$.

Example 7.8

1. SAT $\in$ PCP$(0, n, 0)$. The y value is the satisfying assignment. $M^{()}$ makes all n queries and does not use random bits.

2. If ϕ is a formula let C be the number of clauses in it. 3-SAT $\in$ PCP$(\lg(C), 3, \frac{C-1}{C})$. The y value is the satisfying assignment. M picks a random clause and queries the 3 truth assignments. If they satisfy the clause, output YES, else NO. If $\phi \in$ 3-SAT then the algorithm will return YES. If $\phi \notin$ 3-SAT then the worst case is if y satisfies all but one of the clauses, hence the probability of error is $\leq \frac{C-1}{C}$.

3. Let $L \in \mathsf{N}$. 3-SAT $\in$ PCP$(L \lg(C) + O(1), 3L, \frac{C-L}{C})$. Iterate the proof in part 2 L times.

Arora et. al [5], building on the work of Arora et. al [6], proved the following Theorem.

We omit the proof which is difficult.

Theorem 7.9

1. SAT $\in$ PCP$(O(\log n), O(1), \frac{1}{2})$.

2. For all constants $0 < \epsilon < 1$, SAT $\in$ PCP$(O(\log n), O(1), \epsilon)$. *(This is easily obtained by iterating the protocol from Part 1.)*

3. SAT $\in$ PCP$(O(\log^2 n), O(\log n), \frac{1}{n})$. *(This is easily obtained by iterating the protocol from Part 1.)*

The result SAT $\in$ PCP$(O(\log^2 n), O(\log n), \frac{1}{n})$ is *not* good enough for proving problems hard to approximate. Ajtai-Komlós-Szemerédi [1] and Impagliazzo-Zuckerman [16] improved it by using a technique to reuse random bits by doing a random walk on an expander graph. The next theorem is *not* in their papers; however, one can obtain it from their papers.

See Vazirani [32] (Theorem 29.18).

Theorem 7.10 SAT $\in$ PCP$(O(\log n), O(\log n), \frac{1}{n})$.

8 CLIQ is Hard to Approximate

Arora et. al [5], building on the work of Feige et. al [12], proved that CLIQ is hard to approximate. We will recognize the proof as a gap reduction.

Notation 8.1 If G is a graph then $\omega(G)$ is the size of the largest clique in G.

We state both the upper and lower bound in this proof; however, we only prove the lower bound.

In the following theorem, the size of a graph is the number of vertices.

Theorem 8.2

1. *There is an algorithm that, on input graph G with N vertices, outputs a clique of size $\Omega(\frac{\log^3 N}{N(\log\log N)^2}\omega(G))$. Hence* CLIQ $\in$ POLY-APX. *(Feige [11] proved this. We omit the proof. We use N for the number of vertices since n will be the length of a string in an* NP*-set A.)*

2. *Let A be an* NP*-complete problem. Let c, d be such that there $A \in$* $\mathrm{PCP}(c\lg n, d\lg n, \frac{1}{n})$ *(such a c, d exists by Theorem 7.10). There is a reduction that maps $x \in \Sigma^*$ ($|x| = n$) to a graph G on $N = n^{c+d}$ vertices such that:*
   - *If $x \in A$ then $\omega(G) \geq n^c = N^{c/(c+d)}$.*
   - *If $x \notin A$ then $\omega(G) \leq n^{c-1} = N^{(c-1)/(c+d)}$.*

3. $\mathrm{GAP}(\mathrm{CLIQ}, N^{c/(c+d)}, N^{(c-1)/(c+d)})$ *is* NP*-hard. (This follows from Part 2 and Lemma 6.4.)*

4. *If there is an approximation algorithm for* CLIQ *that, on input G, where G has N vertices, returns a number $> \frac{1}{N^{1/(c+d)}}\omega(G)$, then* P = NP. *(This follows from Part 2 and Lemma 6.4.)*

5. *Assuming* P $\neq$ NP, CLIQ $\in$ POLY-APX $-$ LOG-APX. *(This follows from parts 1,4.)*

Proof:

We just prove part 2.

Let A, c, d be as in the statement of the theorem. To avoid notational clutter we say *run the* PCP *on* (x, τ) rather than give the RPOTM-BA for A a name.

1. Input x of length n.

2. Form a graph $G = (V, E)$ as follows:

   (a) $V = \{0,1\}^{c \lg n + d \lg n}$. Note that there are $N = n^{c+d}$ vertices.

   (b) This step will help us determine the edges. For each vertex write it as $\tau\sigma$ where $|\tau| = c \lg n$ and $|\sigma| = d \lg n$. Run the PCP on (x, τ) and answer the ith query made with the ith bit of σ. Keep track of which queries were made, what the answers were, and if the computation accepted.

   (c) We now determine the edges. Let $\tau\sigma$ and $\tau'\sigma'$ be two vertices. We know both the queries and the answers made when running PCP(x, τ) using σ for the answers, and running PCP(x, τ') using σ' for the answers. Connect the two vertices $\tau\sigma$ and $\tau'\sigma'$ if (1) both represent computations that accept, (2) $\tau \neq \tau'$, and (3) the answers to queries do not contradict.

3. Output the graph.

Note the following:

1. If $x \in A$ then there exists a consistent way to answer the bit-queries such that, for all $\tau \in \{0,1\}^{c \lg n}$, the PCP on (x, τ) accepts. Hence $\omega(G) \geq 2^{c \lg n} = n^c = N^{c/(c+d)}$.

2. If $x \notin A$ then any consistent way to answer the bit-queries will make $\leq \frac{1}{n}$ of the $\tau \in \{0,1\}^{c \lg n}$ accept. Hence $\omega(G) \leq n^{c-1} = N^{(c-1)/(c+d)}$.

▮

Note 8.3 Theorem 8.2 showed that, for $\delta = \frac{1}{c+d}$, if there is an algorithm that returns a number $> \frac{1}{n^\delta}\omega(G)$ then P = NP. Better results are known:

1. Hastad [14] showed that, for all $0 \leq \delta < 1$, if there is an algorithm that returns a number $\geq \frac{1}{n^\delta}\omega(G)$ then ZPP = NP.

2. Zuckerman [34] showed that, for all $0 \leq \delta < 1$, if there is an algorithm that returns a number $\geq \frac{1}{n^\delta}\omega(G)$ then P = NP.

9 SETCOVER is Hard to Approximate

Problem 9.1 SETCOVER
INSTANCE: n and Sets $S_1, \ldots, S_m \subseteq \{1, \ldots, n\}$. QUESTION: What is the smallest size of a subset of S_i's that covers all of the elements in $\{1, \ldots, n\}$?

Notation 9.2 An algorithm *approximates* SETCOVER *within a factor of* $f(n)$ if it outputs a number that is $\le f(n)$ times the optimal.

The following are known.

Theorem 9.3

1. *Chvatal [8] showed that a simple greedy algorithm approximates* SETCOVER *within a factor of* $\ln(n)$.
2. *Slavík [31] gave a slight improvement by replacing the* $\ln n$ *with* $\ln n - \ln(\ln n) - O(1)$.
3. *Lund and Yannakakis [26] showed that, for any* $0 < c < 1/4$, *if* SETCOVER *can be approximated within a factor of* $c \ln n$ *then* $\mathrm{NP} \subseteq \mathrm{DTIME}(n^{\mathrm{polylog}(n)})$.
4. *Raz and Safra [29] showed that there is a constant c such that if* SETCOVER *can be approximated within a factor of* $c \ln n$ *then* $\mathrm{P} = \mathrm{NP}$. *The constant is not explicit in the paper.*
5. *Alon et. al [2] showed that there is a constant c such that if* SETCOVER *can be approximated within a factor of* $c \ln n$ *then* $\mathrm{P} = \mathrm{NP}$.
6. *Dinur and Steurer [10] showed that if* SETCOVER *can be approximated within a* $(1 - o(1)) \ln n$ *then* $\mathrm{P} = \mathrm{NP}$.

Note 9.4 How does m, the number of sets, impact these results? The lower bound proofs apply for m very small, like $n^{0.0001}$. Hence, when we later do reductions of SETCOVER to other problems we can assume m is small.

The lower bound papers do not use PCP's. They instead use a close cousin: 2-prover-1-round interactive proof systems.

10 MAX3SAT is Hard to Approximate

Arora et. al [5] proved that MAX3SAT is hard to approximate. We will recognize the proof as a gap reduction.

Notation 10.1 If ϕ is a 3CNF formula (so every clause has ≤ 3 literals) then MAX3SAT(ϕ) is the max number of clauses that can be satisfied simultaneously.

We state both the upper and lower bound in this proof. We prove one of the upper bounds; however, our main interest is in the lower bound.

Notation 10.2 Let $q \in \mathsf{N}$. Then $C(q)$ is the the maximum number of clauses in a 3CNF formula on 2^q variables. Note that $C(q) = O(2^{3q})$. Since q is constant, $C(q)$ is constant.

In the following theorem, the size of a 3CNF formula is the number of clauses.

Theorem 10.3

1. *Restrict* MAX3SAT *to formulas that have* exactly *three literals per clause. There is an algorithm that, given such a ϕ, returns a number that is* $\ge 0.875\mathrm{MAX3SAT}(\phi)$.

2. *Karloff and Zwick [18] have a randomized polynomial time algorithm for* MAX3SAT *(note–clauses can have 1,2, or 3 literals) that, on input ϕ, does the following: (1) if $\phi \in$* SAT *returns an assignment that satisfies* $\ge 0.875\mathrm{MAX3SAT}(\phi)$ *of the clauses, (2) if $\phi \notin$* SAT *then there is good evidence that the algorithm still returns an assignment that satisfies at least* $0.875\mathrm{MAX3SAT}(\phi)$.

3. *Let A be* NP*-complete. Let $c, q \in \mathsf{N}$ such that* $A \in \mathrm{PCP}(c \lg n, q, 0.25)$ *(such a c, q exists by Theorem 7.9). There is a reduction that maps $x \in \Sigma^*$ to a* 3CNF *formula ϕ such that:*

   (a) If $x \in A$ then $\mathrm{MAX3SAT}(\phi) = |\phi|$. *($\phi \in$* 3-SAT *so all $|\phi|$ clauses are satisfied.)*

   (b) If $x \notin A$ then $\mathrm{MAX3SAT}(\phi) \le \left(1 - \frac{3}{4C(q)}\right)|\phi|$.

(c) The output ϕ has exactly 3 literals per clause.

4. $\mathrm{GAP}(\mathrm{MAX3SAT}, m, \left(1 - \frac{3}{4C(q)}\right)m)$ *is* NP*-hard (where m is the number of clauses). This holds even if ϕ is restricted to having exactly 3 literals per clause. (This follows from Part 3 and Lemma 6.4.)*
5. *If there is an approximation algorithm for* MAX3SAT *that, on input ϕ, returns a number* $> \left(1 - \frac{3}{4C(q)}\right)\mathrm{MAX3SAT}(\phi)$ *then* $\mathrm{P} = \mathrm{NP}$*. (This follows from Part 3 and Lemma 6.4.)*
6. *Assuming* $\mathrm{P} \neq \mathrm{NP}$*,* $\mathrm{MAX3SAT} \in \mathrm{APX} - \mathrm{PTAS}$*. (The problem that separates them is* MAX3SAT *restricted to formulas that have exactly 3 literals per clause. This Part follows from Parts 1 and 4.)*

Proof:

We just prove parts 1,3.

1) We first give a randomized algorithm: Assign each variable to TRUE or FALSE at random. The probability of a particular clause being satisfied is $\frac{7}{8} = 0.875$, so by linearity of expectation we expect $\frac{7}{8}$ of the clauses to be satisfied. This gives a randomized algorithm that outputs a number $\geq 0.875\mathrm{MAX3SAT}(\phi)$. This algorithm can be derandomized using the method of conditional probabilities. Details can be found in either Vazirani's book [32] or Shmoys-Williamson's book [33].

3) Let A, c, q be as in the statement of the theorem. To avoid notational clutter we say *run the* PCP *on* (x, τ) rather than give the RPOTM-BA for A a name.

1. Input x.
2. Form a 3CNF formula ψ as follows:
   (a) The PCP for A can only make $2^{q+c\lg n} = 2^q n^c$ possible bit-queries. There are $2^q n^d$ variables, one for each possible bit-query.
   (b) For every $\tau \in \{0,1\}^{c\lg n}$ do the following. For every $\sigma \in \{0,1\}^q$ run the $\mathrm{PCP}(x, \tau)$ using σ for the query answers. Keep track of which ones accepted and which ones rejected. From this information form a formula on $\leq 2^q$ variables that is T iff the PCP accepts with those answers (using τ for the coin flips). Constructing the

formula takes poly time since q (and hence 2^q) is constant. Convert this formula to 3CNF (this easily takes poly time). Call the result ψ_τ. Note that ψ_τ has between 1 and $C(q)$ clauses.

(c) ψ is the AND of the n^c formulas from the last step. Note that ψ is in 3CNF form.

Note the following.

1. Assume $x \in A$. Then there exists a consistent way to answer the bit-queries such that, for all $\tau \in \{0,1\}^{c \lg n}$, the PCP on (x, τ) accepts. Hence every ψ_τ can be satisfied simultaneously. Therefore the fraction of clauses of ψ that can be satisfied is 1. Hence MAX3SAT$(\phi) = |\phi|$.

2. Assume $x \notin A$. Then every consistent way to answer the bit-queries will make $\le \frac{1}{4}$ of the $\tau \in \{0,1\}^{c \lg n}$ accept. We need to estimate the fraction of clauses of ψ that are satisfied. This fraction is maximized when the following occurs: (1) all n^c of the ψ_τ have $C(q)$ clauses, (2) there is an assignment that satisfies all $C(q)$ clauses in $1/4$ of the ψ_τ, and $C(q) - 1$ clauses in $3/4$ of the ψ_τ. Hence the fraction of clauses satisfied is

$$\frac{(n^c/4)C(q) + (3n^c/4)(C(q)-1)}{n^c C(q)} = \frac{n^c C(q) - (3n^c/4)}{n^c C(q)} = 1 - \frac{3}{4C(q)}$$

Hence MAX3SAT$(\phi) \le \left(1 - \frac{3}{4C(q)}\right)|\phi|$.

▮

Theorem 10.3 shows that if there is an algorithm that returns a number $> (1 - \frac{3}{4(q)})$MAX3SAT(ϕ) then P = NP. Hastad [15] proved the strongest result possible, which we prove (assuming some other hard results) in Theorem 12.6.

11 The Gap Lemmas On Gap Problems

In Theorem 8.2 we did not just prove that CLIQ was NP-hard to approximate, we proved that GAP(CLIQ, $N^{c/(c+d)}$, $N^{(c-1)/(c+d)}$) is NP-hard. In fact, all of the approximation results had as a byproduct of the proof that some GAP problem was hard.

Lemmas 6.4 and 6.7 (the Gap Lemmas) showed how a reduction from an NP-hard set A to a function g implies that the function g is hard to approximate. We rework the Gap Lemmas so that they start with an NP-hard Gap Function. We omit proofs since they are similar to those of the original Gap Lemmas.

Lemma 11.1 *Let $a(n), b(n)$ be functions from N to N such that $\frac{b(n)}{a(n)} < 1$. Let $a'(n), b'(n)$ be functions from N to N such that $\frac{b'(n)}{a'(n)} < 1$. Let f be a max-problem such that $\mathrm{GAP}(f, a'(n), b'(n))$ is NP-hard. Let g be a max-problem. Let $a(n)$ and $b(n)$ be functions from N to N such that (1) $\frac{b(n)}{a(n)} < 1$, and (2) b is computable in time polynomial in n. Assume there exists a polynomial time reduction that maps x to y such that the following occurs:*

- *If $f(x) \geq a'(|x|)$ then $g(y) \geq a(|y|)$.*
- *If $f(x) \leq b'(|x|)$ then $g(y) \leq b(|y|)$.*

Then:

1. *$\mathrm{GAP}(g, a(n), b(n))$ is NP-hard (this follows from the premise).*
2. *If there is an approximation algorithm for g that, on input y, returns a number $> \frac{b(|y|)}{a(|y|)} g(y)$, then $\mathrm{P} = \mathrm{NP}$.*

Lemma 11.2 *Let $a(n), b(n)$ be functions from N to N such that $\frac{b(n)}{a(n)} < 1$. Let $a'(n), b'(n)$ be functions from N to N such that $\frac{b'(n)}{a'(n)} < 1$. Let f be a min-problem such that $\mathrm{GAP}(f, a'(n), b'(n))$ is NP-hard. Let g be a min-problem. Let $a(n)$ and $b(n)$ be functions from N to N such that (1) $\frac{b(n)}{a(n)} < 1$, and (2) b is computable in time polynomial in n. Assume there exists a polynomial time reduction that maps x to y such that the following occurs:*

- *If $f(x) \geq a'(|x|)$ then $g(y) \geq a(|y|)$.*
- *If $f(x) \leq b'(|x|)$ then $g(y) \leq b(|y|)$.*

Then:

1. *$\mathrm{GAP}(g, a(n), b(n))$ is NP-hard (this follows from the premise).*

2. *If there is an approximation algorithm for g that, on input y, returns a number $> \frac{b(|y|)}{a(|y|)}g(y)$, then* $\mathrm{P} = \mathrm{NP}$.

Def 11.3 We will refer to reductions like the ones in Lemma 11.1 and 11.2 as *Gap Reductions with ratio* $\frac{b(n)}{a(n)}$. This is the same terminology we use for reductions like those in Lemma 6.4 and 6.7; however, the meaning will be clear from context.

12 MAX3LIN and MAX3SAT

Everything in this section is due to Hastad [15].

Problem 12.1 MAX3LIN

INSTANCE: A set of linear equations $E = E_1, \ldots, E_m$ in Z_2 where each E_i has 3 variables.

QUESTION: Find an assignment (of 0's and 1's) to the variables to maximize the number of equations that are true.

The decision problem associated to MAX3LIN, *is there some assignment that satisfies all clauses*, is in P by Gaussian Elimination. We now define a Gap version of MAX3LIN which we will later state is NP-hard.

Problem 12.2 ϵ-GAP-MAX3LINT

INSTANCE: A set of linear equations $E = E_1, \ldots, E_m$ in Z_2 where each E_i has 3 variables and we are promised *that one of the following occurs.*

1. $\mathrm{MAX3LIN}(E) \le (\frac{1}{2} + \epsilon)m$.
2. $\mathrm{MAX3LIN}(E) \ge (1 - \epsilon)m$.

QUESTION: Determine which is the case.

Theorem 12.3

1. *There is a randomized algorithm that, given an instance E of* MAX3LIN*, returns a number* $\ge 0.5\mathrm{MAX3LIN}(E)$.
2. *There is an algorithm that, given an instance E of* MAX3LIN*, returns a number* $\ge 0.5\mathrm{MAX3LIN}(E)$.

Exercise 12.4 Prove Theorem 12.3.

The following theorem is proven using the PCP machinery. We omit the rather difficult proof.

Theorem 12.5

1. *If there exists* ϵ *so that* ϵ-GAP-MAX3LINT *is in* P *then* P = NP.
2. *If there exists* ϵ *so that* MAX3LIN *is* $(0.5+\epsilon)$*-approx then* P = NP. *(This follows from Part 1.)*

Proof sketch: This is not a sketch of the proof. This is just two of the many ideas that went into the proof. We warn the reader that we are giving a simplified version of these ideas.
Idea One PCP was defined with 1-sided error. Recall that the definition of PCP is

1. If $x \in A$ then there exists y such that, for all τ with $|\tau| = r(n)$, $M^y(x,\tau)$ accepts. In other words, the probability of acceptance is 1.
2. If $x \notin A$ then for all y at most $\epsilon(n)$ of the τ's with $|\tau| = r(n)$ make $M^y(x,\tau)$ accept. In other words, the probability of acceptance is $\le \epsilon(n)$.

Idea One is to define PCP more generally so that if $x \in A$ there will be a small probability of error. Hence there is 2-sided error.
Idea Two Imagine the following scenario.

1. There are 6 bit queries.
2. If the first three are answered 110 then if the next three bits are 001 the machine will accept, but on any other 3-sequence of answers the machine will reject.
3. If the first three are answered 000 then if the next three bits are 101 the machine will accept, but on any other 3-sequence of answers the machine will reject.
4. On any other sequence of the first three answers, the machine will reject.

In the proof of Theorem 8.2 and 10.3 we would consider the relevant sequences to be 110001 and 000101. But the last three bits are irrelevant. Hence we can consider $\{000, 110\}$ as the only possible answer-bit sequences. If we use this PCP in Theorem 8.2 then the graph we get out of a reduction is smaller. If we use this PCP in Theorem 10.3 then the formula we get out of a reduction is smaller. This allows for better and more precise results. ▮

We use Theorem 12.5 to prove a lower bound on MAX3SAT.

Theorem 12.6

1. *Let $0 < \epsilon < 1$. Let our formulas have $4m$ clauses. Then*

$$\text{GAP}(\text{MAX3SAT}, 4(1-\epsilon)m, 3.5(1+\epsilon)m)$$

is NP*-hard.*

2. *Let $0 < \epsilon < 1$. Let our formulas have m clauses. Then*

$$\text{GAP}(\text{MAX3SAT}, (1-\epsilon)m, 0.875(1+\epsilon)m)$$

is NP*-hard. (This follows from Part 1.) This result holds when* MAX3SAT *is restricted to having exactly 3 literals per clause. Hence this result is a lower bound that matches the upper bound in Theorem 10.3.1.*

3. *Let $0 < \delta < 1$. If there is an approximation algorithm for* MAX3SAT *that, on input ϕ, returns a number $> \big(0.875 + \delta\big)\text{MAX3SAT}(\phi)$ then* P = NP*. (This follows from Part 2.)*

Proof:
We do a reduction from

$$\text{GAP}(\text{MAX3LIN}(1-\epsilon)m, (0.5+\epsilon)m)$$

to

$$\text{GAP}(\text{MAX3SAT}(4(1-\epsilon)m, (3.5+\epsilon)m).$$

Lemma 11.1 then gives the result (for part 2 we need to take ϵ small enough).

1. Input a set of equations $E = E_1, \ldots, E_m$ over Z_2.

2. We will form a 3CNF-formula by replacing every equation by a set of clauses and then taking the AND of all of those clauses.

3. For each $1 \le i \le m$ do the following.

   (a) If E_i is of the form $x + y + z \equiv 0 \pmod 2$ then replace it with the following set of clauses.
   $\neg x \vee \neg y \vee \neg z$
   $\neg x \vee y \vee z$
   $x \vee \neg y \vee z$
   $x \vee y \vee \neg z$.

   (b) If E_i is of the form $x + y + z \equiv 1 \pmod 2$ then replace it with the following set of clauses.
   $x \vee y \vee z$
   $\neg x \vee \neg y \vee z$
   $\neg x \vee y \vee \neg z$
   $x \vee \neg y \vee \neg z$

4. We call the resulting formula ϕ. Output ϕ.

Note that ϕ has $4m$ clauses.
Note the following:

1. Think of ϕ as being a set of clauses in groups of 4, called a 4-group, as the construction indicates. Every assignment will satisfy (1) for some 4-groups, all 4 clauses, and (2) for some 4-groups, 3 clauses. That is, it is impossible to satisfy 0 or 1 or 2 clauses in a 4-group.

2. If MAX3LIN$(E) \ge (1-\epsilon)m$ then MAX3SAT$(\phi) \ge 4(1-\epsilon)m + 3\epsilon = (4-\epsilon)m$.

3. If MAX3LIN$(E) \le (0.5+\epsilon)m$ then MAX3SAT$(\phi) \le 4(0.5+\epsilon)m + 3(0.5-\epsilon)m = (2+4\epsilon)m + (1.5-3\epsilon)m = (3.5+\epsilon)m$.

∎

13 Vertex Cover

We show a lower bound on how well VC can be approximated by using a standard reduction of 3-SAT to VC and using it on a GAP version MAX3SAT.

We first state the upper bound whose proof is folklore and omitted.

Theorem 13.1 *There is an algorithm that will, given a graph G, output a number that is* $\le 2\mathrm{VC}(G)$.

Theorem 13.2 *Let $\delta > 0$. If there is an algorithm that, on input a graph G, returns a number* $\le (1.0625 - \delta)\mathrm{VC}(G)$, *then* $\mathrm{P} = \mathrm{NP}$.

Proof:

We will use the fact that, if G has n vertices, then $\mathrm{IS}(G) = n - \mathrm{VC}(G)$.

Let ϵ be such that $\frac{2.125-\epsilon}{2+\epsilon} > 1.0625 + \delta$. By Theorem 12.6

$$\mathrm{GAP}(\mathrm{MAX3SAT}, (1-\epsilon)m, 0.875(1+\epsilon)m)$$

is NP-hard. We give a gap reduction from this GAP problem to VC with ratio $\frac{2.125-\epsilon}{2+\epsilon} > 1.0625 - \delta$. The lower bound on approximating VC will then follow from Lemma 11.1.

1. Input $\phi = C_1 \wedge \cdots \wedge C_m$, a formula in 3CNF where every clause has exactly 3 literals. We are promised that either

   (1) $\mathrm{MAX3SAT}(\phi) \ge (1-\epsilon)m$, or

   (2) $\mathrm{MAX3SAT}(\phi) \le (0.875+\epsilon)m$.

2. Create a graph G as follows.

   (a) For each C_i we have a vertex for each literal. Hence G has $3m$ vertices.

   (b) Put an edge between every pair of vertices in the same C_i. Put an edge between vertices from different C_i's if they contradict each other.

3. (This is commentary, not part of the algorithm).

   Note the following two cases.

   (1) $\mathrm{MAX3SAT}(\phi) \ge (1-\epsilon)m$, so $\mathrm{IS}(G) \ge (1-\epsilon)m$:

$$\mathrm{VC}(G) = 3m - \mathrm{IS}(G) \leq 3m - (1-\epsilon)m = (2+\epsilon)m.$$

(2) $\mathrm{MAX3SAT}(\phi) \leq (0.875+\epsilon)m$, so $\mathrm{IS}(G) \leq (0.875+\epsilon)m$:

$$\mathrm{VC}(G) = 3m - \mathrm{IS} \geq 3m - (0.875+\epsilon)m = (2.125-\epsilon)m.$$

4. Output G.

▮

Are better lower bounds for approximating VC known? Yes. We state two results.

Theorem 13.3 *Let $\delta > 0$.*

1. *If there is an algorithm that, on input a graph G, returns a number $\leq (1.166\ldots - \delta)\mathrm{VC}(G)$, then $\mathrm{P} = \mathrm{NP}$ (the number is actually $\frac{7}{6}$). Hastad [15] showed this.*
2. *If there is an algorithm that, on input a graph G, returns a number $\leq (1.3606\ldots - \delta)\mathrm{VC}(G)$, then $\mathrm{P} = \mathrm{NP}$ (the number is actually $10\sqrt{5} - 21$). Dinur and Safra [9] showed this.*

Are better lower bounds for approximating VC known? Maybe. There are no NP-hardness result known. However, in Section 17 we will state the *Unique Games Conjecture* from which several lower bounds can be proven, including a lower bound of $2 - \delta$ for approximating VC.

14 Simple example: Tetris

Sidebar 14.1 *BILL TO ERIK AND M: I DO NOT UNDERSTAND THIS TETRIS THING. WE SHOULD EITHER DELETE IT OR EXPAND ON IT*

In Tetris, we can create a gap with $c = n^{1-\epsilon}$ for some $\epsilon > 0$. If we let OPT be the number of lines that can be cleared, then this gap is generated with the YES instance as solving the puzzle correctly, and the NO instance as a maximum-optimization for how many lines can be cleared. Letting ϵ be our tolerance for error, as we increase ϵ, our gap widens, and our related maximum bound for the NO instance decreases.

15 Label Cover, Max Rep, and Min Rep

We define two gap problems, state that they are NP-hard, and then use them to show lower bounds on other problems. The problems are *not* natural; they are a means to an end.

Def 15.1 Let $G = (A, B, E)$ be a bipartite graph. We assume the following.

1. $|A| = |B| = n$.
2. There is a partition of A into sets $A_1, \ldots, A_k$ where $|A_i| = \frac{n}{k}$.
3. There is a partition of B into sets $B_1, \ldots, B_k$ where $|B_i| = \frac{n}{k}$.
4. We create a new bipartite graph as follows.
   (a) The vertices on the left are the A_i's.
   (b) The vertices on the right are the B_i's.
   (c) There is an edge from A_i to B_j if there exists $a \in A_i$ and $b \in B_j$ such that $(a, b) \in E$ (the original edges). These new edges are called *superedges.*
5. A *label cover* is two subsets $A' \subseteq A$ and $B' \subseteq B$.
6. Given a label cover we say that a superedge (A_i, B_j) is *covered* if there exists $a \in A_i \cap A'$, $b \in B_j \cap B'$ such that $(a, b) \in E$. Notice that so far we have not demanded anything of our label covering.

We will put two kinds of demands on our label covering. We state them here informally since the actual problem we use will involve approximations.

Problem 15.2

- *The* MAX REP *problem will insist (1) the label covering picks exactly one vertex from each A_i and from each B_j, (2) The number of covered superedges is maximized.*
- *The* MIN REP *problem will insist that (1) every superedge be covered, (2) the number $|A'| + |B'|$ is minimized.*

Problem 15.3

ϵ-GAP-MAXR.

INSTANCE: A bipartite $G = (A, B, E)$ as in Definition 15.1.

QUESTION: We only look at label covers which take exactly one element from each A_i and each B_j.

We are promised that one of the following occurs.

- *There is such a label covering which covers all superedges.*
- *Every such label covering covers at most an ϵ fraction of the superedges.*

The question is to determine which case happens.

Raz [28] showed the following. The proof uses the PCP machinery and is omitted.

Sidebar 15.4 *BILL TO ERIK AND M: I looked at Raz's paper and I just don't see this theorem there. I am also trying to find an analogous theorem for* MIN REP*, which is why I looked there. I have not found that anywhere either.*

Theorem 15.5 *Let $0 < \epsilon < 1$.*

1. *If ϵ-*GAP-MAXR *is in* P *then* P = NP.
2. *If there is an algorithm that on input an instance G of* MIN REP *returns a number $\ge \epsilon$*MAX REP(G) *then* P = NP. *(This follows from Part 1 and Lemma 6.4.*
3. *If $\frac{1}{2^{\log^{1-\epsilon}}}$-*GAP-MAXR *is in* P *then* NP $\subseteq$ DTIME$(n^{\text{polylog}(n)})$.
4. *If there is an algorithm that on input an instance G of* MIN REP *returns a number $\ge \frac{1}{2^{\log^{1-\epsilon}}}$*MAX REP$(G)$ *then* NP $\subseteq$ DTIME$(n^{\text{polylog}(n)})$. *(This follows from Part 3 and a variant of Lemma 6.4.*

We define the dual problem, MIN REP.

Problem 15.6

ϵ-GAP-MINR.

INSTANCE: A bipartite $G = (A, B, E)$ as in Definition 15.1, and ϵ.

QUESTION: We only look at label covers which cover every superedge. We are promised that one of the following occurs. Let S be the number of superedges.

- *There is label covering of size $2S$ that covers every superedge (this is optimal).*
- *Every label covering that covers every superedge has at least $\frac{1}{\epsilon}$ nodes.*

The question is to determine which case happens.

Sidebar 15.7 *BILL TO ERIK AND M: I assume the theorem below is true but I do not know a reference.*

Theorem 15.8 *Let $0 < \epsilon < 1$.*

1. *If ϵ-GAP-MINR is in P then P = NP.*
2. *If there is an algorithm that on input an instance G of* MIN REP *returns a number $\le \frac{1}{\epsilon}$MIN REP(G) then P = NP. (This follows from Part 1 and Lemma 6.7.*
3. *If $\frac{1}{2^{\log^{1-\epsilon}}}$-GAP-MINR is in P then NP $\subseteq$ DTIME$(n^{\mathrm{polylog}(n)})$.*
4. *If there is an algorithm that on input an instance G of* MIN REP *returns a number $\le 2^{\log^{1-\epsilon}}$MIN REP$(G)$ then NP $\subseteq$ DTIME$(n^{\mathrm{polylog}(n)})$. (This follows from Part 3 and a variant of Lemma 6.7.)*

16 A Reduction from MIN REP

We show that the Directed Steiner Forest problem is hard by reducing MIN REP to it.

Problem 16.1 *Directed Steiner Forest* (DSF*).*

INSTANCE: A weighted directed graph G and a set of ordered pairs of vertices $\{(a_i, b_i)\}$.

QUESTION: Find a subgraph G that has, for all i, a path from a_i to b_i of minimal weight. Note that this subset will be a forest.

Theorem 16.2 *Let $c \ge 1$. If there is an algorithm that, on input an instance of* DSF*, outputs a number that is $\le c \times$ DSF(G), then P = NP.*

Proof: Let ϵ be such that $\frac{1}{\epsilon} = c$. We give a gap reduction from ϵ-GAP-MINR to DSF and then apply Lemma 11.2.

1. Input is a bipartite graph $G = (A, B, E)$, $|A| = |B| = n$, a partition of A into sets $A_1, \ldots, A_k$ where $|A_i| = \frac{n}{k}$, and a partition of B into sets $B_1, \ldots, B_k$ where $|B_i| = \frac{n}{k}$. Let S be the number of superedges. We are told that either (1) there is a label cover of size $2S$ that covers all superedges, or (2) any label cover that covers all the superedges has $\ge \frac{1}{\epsilon}2S$ vertices.

2. Create a directed graph G' as follows.

   (a) All of the vertices and edges of G are present. All of the edges have weight 0.

   (b) For each A_i there is a new vertex a_i. For all $v \in A_i$ there is an edge (a_i, v) of weight 1.

   (c) For each B_i there is a new vertex b_i. For all $v \in B_i$ there is an edge (v, b_i) of weight 1.

   (d) There is also an edge from a_i to b_i of weight w to be determined later.

   (e) The set of ordered pairs is all $\{(a_i, b_j) \colon (A_i, B_j) \text{ is a superedge}\}$.

3. (This is not part of the algorithm. This is commentary.) Let S be the number of superedges. It is easy to see that there is a label cover of size L iff there is a Directed Steiner Forest of weight L. Hence

   - If there is a label cover of size $2S$ that covers all the superedges then there is a Directed Steiner Forest of weight $2S$ which implies that there is a DSF of weight $2S$.
   - If the minimum sized label cover is of size $\ge \frac{1}{\epsilon}2S$ then the optimal DSF has weight $\ge \frac{1}{\epsilon}2S$.

4. Output the instance of MIN REP that you created.

▮

17 The Unique Games Conjecture

Recall that in our definitions of GAP-MAXR the promise is that either (1) there is a label covering with one vertex per A_i and B_j which covers all superedges, or (2) every such label covering covers at most an ϵ fraction of the superedges. What if we relaxed the promise of part (1)? Consider the following gap problem.

Def 17.1 ϵ-2-sided-GAP-MAXR.

INSTANCE: A bipartite $G = (A, B, E)$ that has the vertices partitioned as in Definition 15.1.

QUESTION: We only look at label cover which takes exactly one element from each A_i and each B_j. We are promised that one of the following occurs.

- There is such a label covering which covers fraction $(1 - \epsilon)$ of the superedges.
- Every such label covering covers at most an ϵ fraction of the superedges.

The question is to determine which case happens.

Khot [19] made the following conjecture.

Conjecture 17.2 *The* Unique Games Conjecture (UGC) *is that, for all* $\epsilon > 0$*,* ϵ*-2-sided*-GAP-MAXR *is* NP*-hard. (The name* Unique Games Conjecture *comes from another formulation of it.)*

For more on UGC see Khot's survey [20] and Klarreich's exposition [24]. Is the conjecture true? We give two thoughts.

Argument for UGC

1. UGC has great explanatory power. We give some examples.
    - As noted in Theorem 13.1 and 13.3, (1) there is a poly algorithm that will, given a graph G, return a number $\le 2\text{VC}(G)$, and (2) if P $\neq$ NP then there is no poly algorithm that returns $\ge 1.306\ldots\text{VC}(G)$. Khot and Regev [23] showed the following: If UGC then there is no poly algorithm that returns $\ge (2-\epsilon)\text{VC}(G)$, which matches the upper bound.

- Recall that the MAXCUT problem is as follows: given a weighted graph G, find a partition of the vertices into 2 parts that maximizes the sum of the weights of the edges between the parts. Goemans and Williamson [13] showed that there is an algorithm that will, given a graph G, return a number $\geq 0.878 \cdots \text{MAXCUT}(G)$. The actual number is

$$\alpha = \min_{0 \leq \Theta \leq \pi} \frac{2\theta}{\pi(1-\cos(\theta))}.$$

  Hastad [15] showed that if $\text{P} \neq \text{NP}$ then there is no poly algorithm that returns $\geq 0.941 \cdots \text{MAXCUT}(G)$. (The actual number is $\frac{16}{17}$.) Khot et al. [21] showed the following: If UGC then there is no poly algorithm that returns $\geq (0.9439\ldots+\delta)\text{MAX2SAT}(G)$. This does not match the upper bound; however, it is closer to it than the result obtained by assuming $\text{P} \neq \text{NP}$.

In both the case of Vertex Cover and MAXCUT the proofs of the upper and lower bounds use very different techniques. Hence the fact that from UGC we get a lower bound that just happens to match the lower bound is evidence for UGC. The proofs of the upper and lower bound for MAX2SAT also use very different techniques.

2. Khot et al. [22] proved a weaker version of UGC, called the 2-2 games conjectures. See also the exposition by Klarreich [25].

An argument for why it is false

1. It is possible we will obtain that explanatory power from the assumption $\text{P} \neq \text{NP}$.

2. Arora et al. [4] obtained a subexponential algorithm for ϵ-2-sided-GAP-MAXR. Note that the algorithm is not polynomial and has not been improved on since 2010.

Unlike P vs NP and many other conjectures, the community is truly split on this conjecture.

References

[1] M. Ajtai, J. Komlós, and E. Szemerédi. Deterministic simulation in LOGSPACE. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, pages 132–140. ACM, 1987.
`https://doi.org/10.1145/28395.28410`.

[2] N. Alon, D. Moshkovitz, and S. Safra. Algorithmic construction of sets for k-restrictions. *ACM Transactions on Algorithms*, 2(2):153–177, 2006.
`https://doi.org/10.1145/1150334.1150336`.

[3] S. Arora. Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. *Journal of the Association of Computing Machinery (JACM)*, 45(3):501–555, 2007.
`https://dl.acm.org/doi/10.1145/290179.290180`.

[4] S. Arora, B. Barak, and D. Steurer. Subexponential algorithms for unique games and related problems. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 563–572. IEEE Computer Society, 2010.
`https://doi.org/10.1109/FOCS.2010.59`.

[5] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and the hardness of approximation problems. *Journal of the ACM*, 45(3):501–555, 1998.
`https://doi.org/10.1145/278298.278306`.

[6] S. Arora and S. Safra. Probabilistic checking of proofs: A new characterization of NP. *Journal of the ACM*, 45(1):70–122, 1998.
`https://doi.org/10.1145/273865.273901`.

[7] N. Christofides. Worst case analysis of a new heuristic for the Travelling Salesman Problem, 1976.
`https://apps.dtic.mil/sti/pdfs/ADA025602.pdf`.

[8] V. Chvatal. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 4:233–235, 1979.
`https://www.jstor.org/stable/3689577?seq=1#metadata_info_tab_contents`.

[9] I. Dinur and H. Safra. On the hardness of approximating minimum vertex cover. *Annals of Mathematics*, pages 439–485, 2005. http://www.wisdom.weizmann.ac.il/~dinuri/mypapers/vc.pdf.

[10] I. Dinur and D. Steurer. Analytical approach to parallel repetition. In *Proceedings of the 46th annual ACM Symposium on Theory of Computing*, pages 624–633, 2013. https://dl.acm.org/doi/pdf/10.1145/2591796.2591884.

[11] U. Feige. Approximating maximum clique by removing subgraphs. *SIAM J. Discret. Math.*, 18(2):219–225, 2004.

[12] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy. Interactive proofs and the hardness of approximating cliques. *Journal of the ACM*, 43(2):268–292, 1996. https://doi.org/10.1145/226643.226652.

[13] M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, 42(6):1115–1145, 1995. https://dl.acm.org/doi/pdf/10.1145/227683.227684.

[14] J. Hastad. Clique is hard to approximate within $n^{1-\epsilon}$. *Acta Mathematica*, 1999. https://www.csc.kth.se/~johanh/cliqueinap.pdf.

[15] J. Hastad. Some optimal inapproximability results. *Journal of the Association of Computing Machinery (JACM)*, 48(4):798–859, 2001. https://dl.acm.org/doi/10.1145/502090.502098.

[16] R. Impagliazzo and D. Zuckerman. How to recycle random bits. In *30th Annual Symposium on Foundations of Computer Science, Research Triangle Park, North Carolina, USA, 30 October - 1 November 1989*, pages 248–253. IEEE Computer Society, 1989. https://doi.org/10.1109/SFCS.1989.63486.

[17] A. Karlin, N. Klein, and S. Oveis-Gharan. A (slightly) improved approximation algorithm for metric TSP, 2020. https://arxiv.org/abs/2007.01409.

[18] H. Karloff and U. Zwick. A 7/8-approximation algorithm for MAX 3-SAT? In *38th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 406–415. IEEE Computer Society, 1997. `http://www.cs.umd.edu/~gasarch/BLOGPAPERS/max3satu.pdf`.

[19] S. Khot. On the power of unique 2-prover 1-round games. In *Proceedings of the 17th Annual IEEE Conference on Computational Complexity, Montréal, Québec, Canada, May 21-24, 2002*, page 25. IEEE Computer Society, 2002. `https://doi.org/10.1109/CCC.2002.1004334`.

[20] S. Khot. On the unique games conjecture (invited survey). In *Proceedings of the 25th Annual IEEE Conference on Computational Complexity, CCC 2010, Cambridge, Massachusetts, USA, June 9-12, 2010*, pages 99–121. IEEE Computer Society, 2010. `https://doi.org/10.1109/CCC.2010.19`.

[21] S. Khot, G. Kindler, E. Mossel, and R. O'Donnell. Optimal inapproximability results for MAX-CUT and other 2-variable CSPs? *SIAM Journal on Computing*, 37(1):319–357, 2007. `https://www.cs.cmu.edu/~odonnell/papers/maxcut.pdf`.

[22] S. Khot, D. Minzer, and M. Safra. Pseudorandom sets in Grassmann graph have near-perfect expansion. In M. Thorup, editor, *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 592–601. IEEE Computer Society, 2018. `https://doi.org/10.1109/FOCS.2018.00062`.

[23] S. Khot and O. Regev. Vertex cover might be hard to approximate to within $2 - \epsilon$. In *18th Annual IEEE Conference on Computational Complexity (Complexity 2003), 7-10 July 2003, Aarhus, Denmark*, page 379. IEEE Computer Society, 2003. `https://doi.org/10.1109/CCC.2003.1214437`.

[24] E. Klarreich. Approximately hard: The unique games conjecture, 2011. This is on the Simons Institute Website.

[25] E. Klarreich. First big steps towards proving the unique game conjecture. *Quanta*, 2018.

[26] C. Lund and M. Yannakakis. On the hardness of approximating minimization problems. *Journal of the ACM*, 41(5):960–981, 1994.
`https://dl.acm.org/doi/pdf/10.1145/185675.306789`.

[27] J. Mitchell. Guillotine subdivisions approximate polygonal subdivisions: a simple polynomial-time approximation scheme for geometric TSP, k-MST, and related problems. *SIAM Journal of Computing*, 28(4):1298–1309, 1999.
`https://epubs.siam.org/doi/abs/10.1137/S0097539796309764`.

[28] R. Raz. A parallel repetition theorem. *SIAM Journal on Computing*, 27(3):763–803, 1998.
`https://dl.acm.org/doi/10.1145/225058.225181`.

[29] R. Raz and S. Safra. A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP. In F. T. Leighton and P. W. Shor, editors, *Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing, El Paso, Texas, USA, May 4-6, 1997*, pages 475–484. ACM, 1997.
`https://doi.org/10.1145/258533.258641`.

[30] A. Serdyukov. On some extremal walks in graphs (in Russian). *Upravlyaemye Sisetmy*, 17:76–79, 1978.
`http://nas1.math.nsc.ru/aim/journals/us/us17/us17_007.pdf`.

[31] P. Slavík. A tight analysis of the greedy algorithm for set cover. *J. Algorithms*, 25(2):237–254, 1997.
`https://doi.org/10.1006/jagm.1997.0887`.

[32] V. Vazirani. *Approximation Algorithms*. Springer, 2001.

[33] D. Williamson and D. Shmoys. *Design of Approximation Algorithms*. Cambridge University Press, 2011.

[34] D. Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. *Theory of Computing*, 3(6):103–128, 2007.
`http://theoryofcomputing.org/articles/v003a006/index.html`.