IAS/Park City Mathematics Series
Volume 00, 0000

Exploring Complexity through reductions

Sanjeev Arora

Since we have few techniques for proving strong lowerbounds on Turing machine computations, an interim approach is to study interrelationships between various complexity questions. We usually prove interrelationships using *reductions*. Examples of such efforts include (in addition to the ones studied in the next three lectures) the following.

(1) Most of cryptography, like message authentication, digital signatures, pseudorandom generation, etc. is based on reductions.
(2) Papadimitriou's complexity class based on non-constructive math proofs [**Pap94**].
(3) Sahai-Vadhan's complete problems for zero-knowledge [**SV97**].
(4) Levin's theory of average case complexity [**Lev**].
(5) Fixed parameter tractability theory introduced by Downey and Fellows [**DF95**].

The next three lectures show this reduction-based methodology in action for three important issues.

We note that there are many examples of complexity classes for which the reduction-based effort has not proved as successful, such as BPP, NP $\bigcap$ coNP, and quantum classes. There are inherent obstacles in proving that these classes have complete problems [**Pap94b**].

LECTURE 1

PCP Theorem and Hardness of Computing Approximate Solutions

This first talk introduces characterization of NP in terms of Probabilistically Checkable Proofs and their application to prove inapproximability results for various optimization problems.

[1]Department of Computer Science, Princeton University, NJ 08544. These notes were scribed by Subhash Khot.
E-mail address: arora@cs.princeton.edu.

©0000 American Mathematical Society

1. Approximation Algorithms

Many optimization problems occuring in practice are NP-hard to solve exactly. For instance,

(1) MAX-3SAT : Given a $3SAT$ formula ϕ, find an assignment that satisfies the maximum number of clauses.
(2) MAX-INDEP-SET : Given a graph G, find the maximum independent set in it.
(3) MIN-COLORING : Given a graph G, find a coloring (i.e. an assignment of colors to vertices such that any two adjacent vertices have different color) that minimizes the number of colors used.

Since we cannot hope to solve these problems exactly, we try to find an approximate solution which is guaranteed to be "close" to the optimal solution.

Definition 1. *An algorithm A achieves an approximation ratio $\alpha \geq 1$ for a maximization (minimization) problem Π if for every instance $I \in \Pi$, the solution $A(I)$ given by the algorithm satisfies $A(I) \geq OPT(I)/\alpha$ ($A(I) \leq \alpha \cdot OPT(I)$) where $OPT(I)$ is the optimum solution to instance I.*

Note that designing such algorithms is nontrivial: the algorithm must be able to find a good upper (lower) bound on $OPT(I)$, even though computing the optimum itself is NP-hard. Note also that the approximation algorithm has to work as promised on *all* inputs.

The field of approximation algorithms has two strands, the first is to find algorithms with good approximation ratio and second, to prove that no good approximation algorithm exists (assuming $P \neq NP$). For example there is a 8/7-approximation algorithm for MAX-3SAT (thanks to a line of work beginning with Johnson [**Joh74**] and ending with [**KZ97**]). A recent result by Håstad [**Hå97**] shows that no better algorithm exists if $P \neq NP$.

Theorem 2. *If there is a $\delta > 0$ such that an approximation ratio $(8/7 - \delta)$ can be achieved for MAX-3SAT in polynomial time then $P = NP$.*

We will discuss results of this second type, i.e. proving inapproximability results.

Let us first consider how to prove a result like Theorem 2. Let us consider a weaker result for MAX-3SAT, namely, that there is some $\epsilon > 0$ such that approximating MAX-3SAT with ratio $(1 + \epsilon)$ is NP-hard.

One way to prove such a result would be come up with a polynomial-time reduction f—let us call it a *magic reduction* in this discussion— from any language $L \in NP$ to $3SAT$ such that

$$x \in L \Longrightarrow MAX \cdot 3SAT(f(x)) = 1$$

$$x \notin L \Longrightarrow MAX \cdot 3SAT(f(x)) < 1 - \epsilon$$

where 1 and $1 - \epsilon$ refer to the maximum fraction of clauses that can be satisfied.

Clearly, if the magic reduction exists for every language $L \in NP$, then we can conclude that achieving an approximation ratio $\frac{1}{1-\epsilon}$ for MAX-3SAT allows us to decide membership in every NP language.

So, does the magic reduction exist? We will see that the classical Cook-Levin technique fails to give such a reduction. However the following more recent results guarantee existence of magic reductions.

Theorem 3. *([**ALMSS92**]) Magic reduction exists for some $\epsilon > 0$ iff*

$$NP = PCP(O(\log n), O(1)).$$

Theorem 4. *([**ALMSS92**], [**AS92**])*

$$NP = PCP(O(\log n), O(1)).$$

These results will be proven later in the course; this lecture will just give an introduction.

2. Probabilistically Checkable Proofs

This section defines the class PCP. First, we recall the classical definition of NP and the Cook-Levin technique for reductions.

NP consists of precisely those languages which have deterministically checkable polynomial size proofs.

$L \in NP$ iff there is a polynomial time TM M such that

$$x \in L \Longrightarrow \exists \Pi \; M(x, \Pi) \; = \; Accept$$
$$x \notin L \Longrightarrow \forall \Pi \; M(x, \Pi) \; = \; Reject$$

2.1. Cook-Levin Revisited

Recall the classical reduction of NP language L to $3SAT$. The reduction uses the idea of a tableau (i.e., transcript) of computation of a NTM M on input x such that $x \in L$ iff there exists a valid tableau (Figure 1).

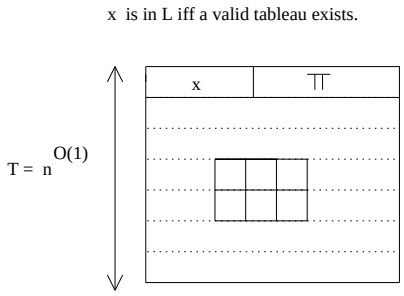

Figure 1. Cook-Levin reduction

The main observation is that the tableau represents a correct computation iff all 2×3 *windows* look allright, i.e. thay satisfy some local consistency conditions. Cook-Levin reduction encodes these consistency checks with $3CNF$ clauses so that a valid tableau exists iff the set of all these $3CNF$ clauses is satisfiable.

But this reduction is not the desired magic reduction. The reason is that when $x \notin L$, we can still satisfy almost all the clauses in the 3CNF formula. Just consider a tableau which in the 2nd line switches to an accepting configuration and stays in this configuration thereafter. The consistency conditions are violated only along the first row of the tableau. Since there are $n^{O(1)}$ rows, the fraction of unsatisfied clasues in the corresponding $3SAT$ formula is $1/n^{O(1)}$, which is much less than any fixed constant ϵ, as required for the magic reduction. Thus the Cook-Levin reduction cannot produce a "gap". We need a more robust representation of the computation than the tableau, which is either correct or looks incorrect almost everywhere. PCPs allow us to do exactly that.

2.2. PCPs

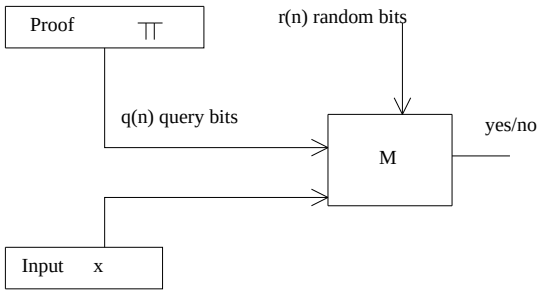

Figure 2. PCP verifier

Definition 5. *The class $PCP(r(n), q(n))$ is defined as the class of all languages for which there exists a probabilistic polynomial time TM M, called the verifier, such that*

- *M has random access to a proof Π.*
- *On input of length n, M uses $r(n)$ random bits and queries $q(n)$ bits from the proof.*
- *Based on the bits read from the proof, M decides whether to accept or reject the input.*

The verifier satisfies following conditions :

$$(completeness)\ x \in L \Longrightarrow \exists \Pi, \quad Pr[M(x,\Pi) = Accept] = 1$$
$$(soundness) \quad x \notin L \Longrightarrow \forall \Pi, \quad Pr[M(x,\Pi) = Accept] \leq 1/2$$

It is easy to see that if a magic reduction exists, then $NP \subseteq PCP(O(\log n), O(1))$. For $L \in NP$, the verifier first converts an input x to a $3SAT$ formula such that if $x \in L$, the formula is satisfiable and if $x \notin L$, at most $(1-\epsilon)$ fraction of the clauses are satisfiable. The verifier expects as a proof a satisfying assignment to this $3SAT$ formula. The verification of the proof consists of picking up a random clause, querying the values of 3 variables in this clause, and accepting iff these values satisfy the clause. The completeness and soundness of the verifier follow from the gap produced by the magic reduction. The soundness is $(1-\epsilon)$ as opposed to $1/2$ as in the definition of PCP, but it can be amplified using a constant number of repeatitions. The other direction of Theorem 3 is left as an exercise; the next section gives a hint how to proceed.

3. Håstad's PCP and Inapproximability of MAX-3SAT

Håstad [**Hå97**] constructs a PCP-verifier for $3SAT$ which reads only 3 bits from the proof.

Theorem 6. *For every $\epsilon > 0$, there is a PCP-verifier M for $3SAT$ that reads a random string r of length $O(\log n)$ and computes query positions i_r, j_r, k_r in the proof Π and a bit b_r. It accepts iff*

$$\Pi[i_r] \oplus \Pi[j_r] \oplus \Pi[k_r] = b_r \tag{1}$$

The verifer has completeness $1-\epsilon$ and soundness $1/2+\epsilon$, i.e.

$$x \in 3SAT \Longrightarrow \exists \Pi \quad Pr_r[M(x,\Pi) = Accept] \geq 1-\epsilon$$
$$x \notin 3SAT \Longrightarrow \forall \Pi \quad Pr_r[M(x,\Pi) = Accept] \leq 1/2+\epsilon.$$

Definition 7. *Let MAX-3LIN-EQN be a problem where input is a system of linear equations mod 2, each equation contaning at most 3 variables, and the problem is to find an assignment to variables that satisfies maximum number of equations.*

Håstad's PCP construction immediately gives the following inapproximability result for MAX-3LIN-EQN.

Theorem 8. *It is NP-hard to approximate MAX-3LIN-EQN with ratio $(2-\delta)$ for any $\delta > 0$.*

Proof. We can construct a system of linear equations from Håstad's PCP verifier. There is a variable for each bit in the proof. For each choice of random string r, there is a linear constraint given by (1). The set of all these constraints is an instance of MAX-3LIN-EQN where either at least $(1-\epsilon)$ fraction of the constraints are satisfiable or at most $(1/2+\epsilon)$ fraction of the constraints are satisfiable. The theorem follows.

Converting each linear constraint into four $3CNF$ clauses, we also get $(8/7-\delta)$-inapproximability result for MAX-3SAT for any $\delta > 0$. □

□

4. Inapproximability of MAX-3SAT(13)

Inapproximability results for many optimization problems can be proved by a "gap preserving" reduction from MAX-3SAT. In this section we prove inapproximability of a restricted version of MAX-3SAT called MAX-3SAT(13) in which every variable occurs in at most 13 clauses. In this case, the number of clauses is linear in the number of variables. We use constuction of expander graphs due to [**LPS86**].

Definition 9. *A graph $G(V,E)$ is called an expander graph if it is d-regular, (d is a constant) and*

$$\forall S \subseteq V, |S| \leq |V|/2, \quad |E(S,\overline{S})| > |S|$$

Here $E(S,\overline{S})$ is the set of all edges with one endpoint in S and other in $\overline{S}$.

Theorem 10. *(* [**LPS86**]*) There is an explicit construction that for every integer k, constructs in poly(k) time a 6-regular expander graph G_k on k vertices.*

We use this expander construction to prove :

Theorem 11. *There is a polynomial time reduction f from MAX-3SAT to MAX-3SAT(13) such that if ϕ is satisfiable, so is $f(\phi)$ and if at most $1-\epsilon$ fraction of clauses in ϕ are satisfiable then at most $1-\frac{1}{19}\epsilon$ fraction of clauses in $f(\phi)$ are satisfiable. Hence MAX-3SAT(13) is hard to approximate with ratio $1+\delta$ for some $\delta > 0$.*

Proof. Suppose a MAX-3SAT instance has n variables $x_1, \ldots, x_n$ and m clauses. Let m_i be the number of clauses in which variable x_i appears. Let $M = \sum_i m_i$. Since every clause contains at most 3 literals, $M \leq 3m$.

For each variable x_i, let $x_i^1, \ldots, x_i^{m_i}$ be m_i new variables and replace the j^{th} occurrence of variable x_i by x_i^j. To ensure that any assignment assigns the same value to $x_i^1, \ldots, x_i^{m_i}$, we construct an expander graph G_{m_i} on m_i vertices given

by Theorem 10. For $1 \leq k < l \leq m_i$, if (k, l) is an edge in the expander graph G_{m_i}, add clauses $(x_i^k \vee \neg x_i^l)$ and $(\neg x_i^k \vee x_i^l)$. These clauses together just say that x_i^k and x_i^l are assigned the same value. The new formula has $6M + m$ clauses and every variable occurs in exactly 12 new clauses and 1 old clause. Clearly, if the old formula was satisfiable, so is the new formula.

On the other hand, assume at most $1 - \epsilon$ fraction of clauses in the old formula are satisfiable. Let σ be an assignment that satisfies maximum number of clauses in the new formula. We claim that σ must assign the same value to all variables $x_i^1, \ldots, x_i^{m_i}$. If this is not the case, partition these variables into sets S and $\overline{S}$ according to the value assigned by σ, with $|S| \leq |\overline{S}|$. By the expansion property, the number of egdes between S and $\overline{S}$ is more than $|S|$. These egdes represent unsatisfied clauses. Complementing the value of all variables in S, we satisfy all these clauses and stop satisfying at most $|S|$ (old) clauses in which the variables in S appear. This gives an assignment satisfying more clauses than σ, a contradiction.

Since all variables $x_i^1, \ldots, x_i^{m_i}$ are assigned the same value, σ induces an assignment for the old formula and can satisfy at most $1 - \epsilon$ fraction of the old clauses. Thus σ satisfies at most $6M + (1 - \epsilon)m$ clauses which is at most a $1 - \frac{1}{19}\epsilon$ fraction of the total number of clauses in the new formula. □

□

5. Inapproximability of MAX-INDEP-SET

Theorem 12. *It is NP-hard to approximate MAX-INDEP-SET with ratio* $1 + \delta$ *for some* $\delta > 0$.

Proof. We reduce MAX-3SAT(13) instance ϕ to a MAX-INDEP-SET instance G (Figure 3). We construct a triangle with vertices x_i, x_j, x_k for every clause of ϕ containing literals x_i, x_j, x_k. We connect every pair $x_i, \overline{x_i}$ occurring in two different clauses. If n is the number of variables in ϕ, the number of clauses is $O(n)$ and the number of vertices in G is also $O(n)$.

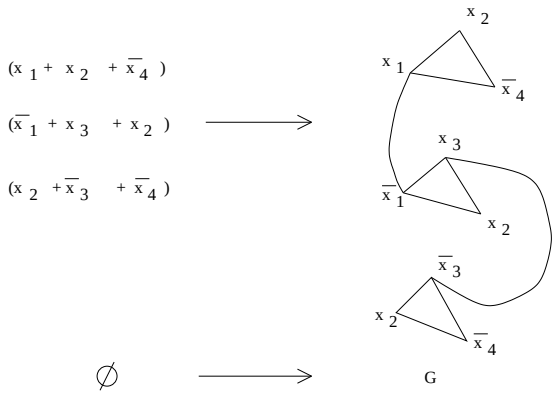

Figure 3. Reduction from MAX-3SAT(13) to MAX-INDEP-SET

It is easy to verify that there is an assignment to ϕ satisfying k clauses iff there is an independent set in G of size k. Since MAX-3SAT(13) is hard to approximate with ratio $1 + \delta$ for some $\delta > 0$ (Theorem 11), so is MAX-INDEP-SET. □

□

Let us denote the size of maximum independent set in graph G by $\alpha(G)$. We can actually conclude that MAX-INDEP-SET is hard to approximate within any arbitrarily large constant factor. This is because for any graph $G(V, E)$, we can construct the graph product G^k, which is a graph on vertex set V^k, and vertices $(u_1, \ldots, u_k)$ and $(v_1, \ldots, v_k)$ are connected iff at least one of the pairs $(u_1, v_1), \ldots, (u_k, v_k)$ is an edge in G. Clearly $\alpha(G^k) = \alpha(G)^k$. This implies that if $\alpha()$ is approximable within factor C, it is also approximable within factor $C^{1/k}$ for any $k \geq 1$. Since we have proved that $\alpha()$ is hard to approximate within $1 + \delta$ for some $\delta > 0$, it follows that it is hard to approximate within any constant factor.

It turns out that MAX-INDEP-SET is extremely hard to approximate, the strongest result is due to Håstad [**Hå96**].

Theorem 13. *$\forall \epsilon > 0$, approximating MAX-INDEP-SET with ratio $n^{1-\epsilon}$ is NP-hard.*

6. Inapproximability of other problems

Inapproximability results are known for various other problems as well. Reductions used to prove these results are shown in Figure 4.

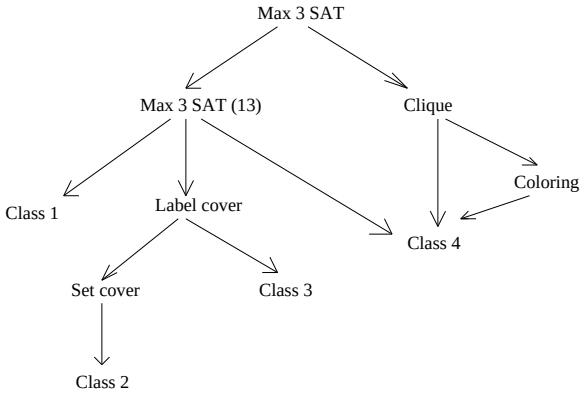

Figure 4. Web of reductions

It is an interesting fact that all these results fall into one of the following 4 classes :

- **Class 1 :** Problems inapproximable within $(1 + \delta)$ for some $\delta > 0$. This class includes MAX-3SAT, MAX-3SAT(13), Vertex cover, Max-Cut, Metric travelling salesman, Steiner trees etc.
- **Class 2 :** Problems inapproximable within $O(\log n)$. For example, Set cover, Dominating set, Hitting set, Hypergraph traversal etc.
- **Class 3 :** Problems inapproximable within $2^{\log^{\delta} n}$ for some $\delta > 0$. For example, Label cover, Nearest lattice vector, Nearest codeword, Quadratic programming etc.
- **Class 4 :** Problems inapproximable within n^{δ} for some $\delta > 0$. For example, MAX-INDEP-SET, Max-Clique, Coloring, Clique-cover etc.

It would be interesting to give some convincing explanation for whether these classes are natural, and whether most interesting open problems fall in one of these classes.

7. History

We give a very brief history of the research that eventually led to the discovery of the PCP theorem. The roots of PCP theorem go back to the Interactive Proofs introduced by Goldwasser, Micali and Rackoff [**GMR89**] and Arthur-Merlin games introduced by Babai [**Bab85**]. Both these proof systems feature a polynomial time probabilistic verifier interacting with an all-powerful adversary called prover. Techniques from program checking due to Blum and Kannan [**BK89**], Lipton [**Lip89**] and Blum, Luby and Rubinfeld [**BLR90**], as well as some ideas about how to represent logical formulae with polynomials (Lund, Fortnow, Karloff and Nisan [**LFKN92**], and Shamir [**Sha92**]) were used to show the surprising results that $IP = PSPACE$ ([**LFKN92**], [**Sha92**]) and $MIP = NEXPTIME$ (Babai, Fortnow and Lund [**BFL91**]). The result $MIP = NEXPTIME$ was "scaled down" to $NP \subseteq PCP(\log n \log\log n, \log n \log\log n)$ by Feige, Goldwasser, Lovász, Safra and Szegedy [**FGLSS91**]. They also observed the connection between PCPs and inapproximability. The PCP theorem was proved by Arora, Safra [**AS92**] and Arora, Lund, Motwani, Sudan and Szegedy [**ALMSS92**].

LECTURE 2
Which problems have strongly exponential complexity?

Let SE (short for *SubExponential*) be the class of search problems that have $2^{o(n)}$ time algorithms where n is a natural way to measure the input size. In an effort to understand which problems are in this class, Impagliazzo, Paturi and Zane [**IPZ98**] define a *subexponential reduction family* or SERF and exhibits a set of problems which are reducible to each other via SERF reductions. This implies that any of these problmes has a subexponential time algorithm iff all of them do. Here we give a brief account of this result as a case study of how reductions can be used to study complexity.

1. SERF-reductions

The first issue to understand is that the class SE, unlike P, is sensitive to the choice of the input parameter. For example, if a 3-SAT formula has n variables and m clauses, what should we think of as a subexponential time algorithm : one that runs in time $2^{o(n)}$ or $2^{o(m)}$? Since w.l.o.g. $m \geq n$ and the trivial algorithm for 3-SAT runs in time 2^n, we must prefer $2^{o(n)}$. Similarly we take the number of vertices (and not the number of edges) as the input parameter for graph problems like INDEPENDENT SET or 3-colorability.

With this choice of parameters, it becomes a nontrivial task to show that these problems are equivalent from the point of view of being computable in subexponential time. The usual polynomial time reduction from 3-SAT to INDEPENDENT SET builds a *graph gadget* for every clause in the 3-SAT formula. For a 3-SAT formula with n variables and m clauses, the reduction gives a graph with $O(m)$ vertices. Note that m can be as large as $\Omega(n^3)$. Thus even if INDEPENDENT SET were in SE, this reduction would only give the trivial conclusion that 3-SAT is in $DTIME(2^{o(n^3)})$. Clearly we want a reduction that increases the parameter of the problem only by a constant factor. On the other hand, we can relax the time complexity of the reduction and allow the reduction to run in time $2^{o(n)}$.

We formally define SERF-reduction next. Recall that a machine M with oracle B, denoted M^B, is a machine that can ask the oracle an answer to any instance of problem B in one unit of time (more precisely, in time linear in the length of the question).

Definition 14. *Problem A is SERF-reducible to problem B, denoted $A \leq_{SERF} B$, if there is some constant C such that for every $\epsilon > 0$, there is a TM $M = M_\epsilon$ such that*

(1) *M^B (M with oracle B) solves A.*
(2) *M runs in time $2^{\epsilon \|x\|_A}$ on input x.*
(3) *Every query x' made by M satisfies $\|x'\|_B \leq C \ \|x\|_A$.*

where $\| \ \|_A$ and $\| \ \|_B$ denote the parameter values for problems A and B respectively.

It is clear that if $A \leq_{SERF} B$ and $B \in DTIME(2^{o(n)})$ then $A \in DTIME(2^{o(n)})$. Also, reducibility is transitive: if $A \leq_{SERF} B$ and $B \leq_{SERF} C$ then $A \leq_{SERF} C$. Thus we are ready to build up a set of complete problems.

2. The Main Theorem

Let k-SAT(c_k) be a restricted version of k-SAT where each literal appears in at most c_k clauses. We will prove that

Theorem 15. $\forall k \geq 3, \quad \exists \ c_k \in \mathbf{N}$ *such that*

$$kSAT \ \leq_{SERF} \ kSAT(c_k).$$

Note that the number of clauses in a k-SAT(c_k) instance is linear in the number of variables. So we can SERF-reduce k-SAT(c_k) to INDEPENDENT SET and 3-Colorability using standard reductions. Indeed, Impagliazzo et al. use k-SAT(c_k) as their "canonical" problem while doing reductions.

The proof of Theorem 15 recalls a basic technique for solving search problems: backtracking. To solve a k-SAT instance ϕ, this technique picks a variable appearing in ϕ, search for a solution that sets this variable to TRUE and if the search fails, backtrack and search for a solution that sets this variable to FALSE. A backtracking algorithm can be viewed as a tree of computation in a natural way.

The proof is of Theorem 15 inspired by a more general form of backtracking that, instead of picking single variables, picks a clause like $x_i \vee x_j \vee x_k \vee x_l$ and branches according to whether $x_i \vee x_j$ is satisfied or $x_k \vee x_l$ is satisfied. In either case, "branching" means we add the respective clause-fragment to the set of clauses we are currently working with and proceed.

Of course, backtracking takes exponential time in the worst case. However, we do not seek to actually solve the instance; just to reduce to k-SAT(c_k). So the proof will use limited backtracking: instead of proceeding all the way to the leaves, it will stop as soon as the formula is an instance of k-SAT(c_k).

Let $\phi = C_1 \wedge \ldots \wedge C_r$ be a k-SAT formula. We say that $C_i \subset C_j$ if the set of literals in clause C_i is a proper subset of the set of literals in clause C_j. In this case, the clause C_j is redundant and we remove it from ϕ, saying clause C_i has *eliminated* clause C_j. Let $Simplify(\phi)$ be a procedure which removes all redundant clauses from ϕ.

Definition 16. *A sunflower is a set of clauses $C_1, \ldots, C_t$ of equal size such that $H = C_1 \bigcap C_2 \ldots \bigcap C_t$ is nonempty. H is called the center and $\{C_1 \setminus H, \ldots, C_t \setminus H\}$ are called the petals of the sunflower. Each petal has the same size ($= |C_i| - |H|$). The size of the sunflower is the number of clauses t in it.*

The reduction (see Figure 2) from k-SAT to k-SAT(c_k) works as follows : Suppose ϕ is a formula we are currently working with. First we simplify ϕ removing all

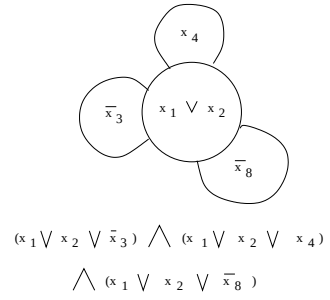

Figure 1. A Sunflower

redundant clauses. Then we pick up a "large" sunflower $\{C_1, \ldots, C_t\}$ with center H. We can satisfy either the center or satisfy all the petals, so we branch out as per the case adding the center or the petals to formula ϕ, and repeat. If there is no large sunflower in ϕ, it means every literal appears in only constantly many clauses and ϕ is a k-SAT(c_k) instance. We can invoke oracle for k-SAT(c_k) to check whether ϕ is satisfiable. To prove that this is a SERF-reduction, we show that the reduction can be made to run in time $2^{\epsilon n}$ for any $\epsilon > 0$. We do so by bounding the number of leaves in the computation tree by $2^{\epsilon n}$.

Let us fix some parameters : α is chosen so that $\alpha / \log \alpha > 4k2^k/\epsilon$. $\theta_0 = 1$ and for $i = 1, \ldots, k-1$, $\beta_i = (4\alpha)^{2^{i-1}-1}$, $\theta_i = \alpha\beta_i$. Finally $c_k = \sum_{i=0}^{k-1} \theta_i$. The algorithm for solving k-SAT instance ϕ with k-SAT(c_k) oracle is shown below. The computation tree is shown in Figure 3. The algorithm traverses the computation tree in depth-first order.

```
Reduce(φ) {
    Simplify(φ).
    For j = 2 to k do {
        For i = 1 to j-1 do {
            If there is a sunflower of size ≥ θ_i, with each
            clause of size j and each petal of size i then {
                Reduce(φ ∧ center).
                Reduce(φ ∧ petals).
                Return.
            }
        }
    }
    If no large sunflower found, ask the k-SAT(c_k) oracle whether
    φ is satisfiable.
    If oracle answers ''yes'' then output ''yes'' and Exit.
    If oracle answers ''no'' then Return.
}
```

Consider a formula ϕ at a leaf of the computation tree. Since ϕ is a leaf, it cannot have any large sunflower in it, and every literal appears in at most $\theta_{i-1} - 1$ clauses of size i. This implies that every literal appears in at most $c_k = \sum_i \theta_i$ clauses. In order to bound the number of leaves, we show that the depth of the tree is at most $\beta_{k-1}n$ and on any path from the root to the leaf, the number of petal-addition steps is at most $\alpha k/n$. Since every path is uniquely determined by the sequence of center/petal-additions, it is clear that the number of leaves is bounded

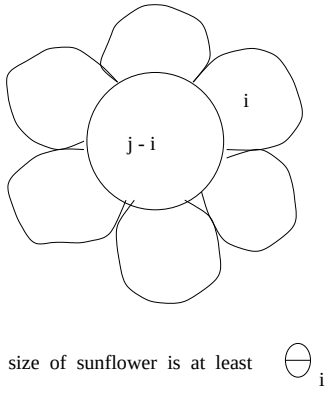

Figure 2. Selecting a large sunflower

by $\sum_{r=0}^{\alpha k/n} \binom{\beta_{k-1} n}{r}$ which we bound by $2^{\epsilon n}$. We prove Theorem 15 in several steps. We assume that every node of the tree contains a formula ϕ that the algorithm is currently working on.

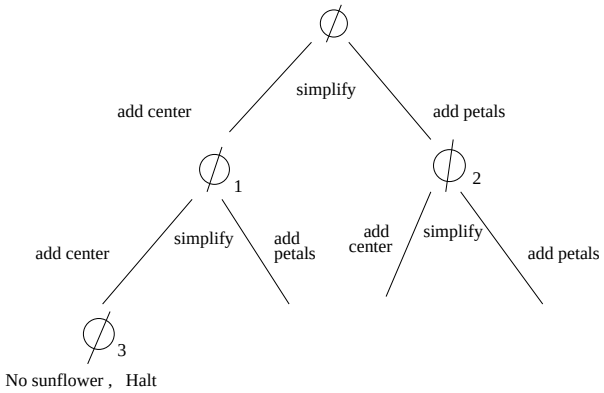

Figure 3. SERF reduction from k-SAT to k-SAT(c_k): the computation tree

Lemma 17. *In any formula ϕ such that a clause of size i has been added to it (as a center or petals) or to any of its predecessors in the path, every literal appears in at most $2\theta_{i-1}$ clauses of size i.*

Proof. It suffices to prove the lemma when ϕ is obtained by adding clause(s) of size i to its parent ϕ' (otherwise consider the most recent predecessor to which an i-clause is added and no more i-clauses are added after that). Consider the sunflower in ϕ' used to add clause(s). The sunflower must consist of clauses of size $j > i$. Looking at the order in which the algorithm executes the loop, there is no sunflower of i-clauses in ϕ' with θ_{i-1} petals of size $i-1$. So each literal appears in at most $\theta_{i-1} - 1$ clauses of size i in ϕ'. The formula ϕ is created by either adding center or the petals to ϕ'. If petals are added, no literal x is contained in more than $\theta_{i-1} - 1$ petals (otherwise ϕ' would contain a sunflower of j-clauses with size θ_{i-1} and petal size $i-1$). Thus no literal x is contained in more than $2\theta_{i-1} - 2$ i-clauses in ϕ. If center is added, it is the only clause added and the lemma still holds. □

□

Lemma 18. *At most $2\theta_{i-1}$ of the added clauses of size i can be eliminated by a single clause C. (Recall that C eliminates a clause C' during simplification step if $C \subset C'$).*

Proof. Observe that every formula ϕ is a "refinement" of its parent ϕ' in the sense that for every clause $D \in \phi$, there is a clause $D' \in \phi'$ such that $D \subseteq D'$. This implies that all clauses ever eliminated by a clause C are eliminated in a single simplification step. If C eliminates an added clause of size i, by lemma 17, no literal appears in more than $2\theta_{i-1}$ clauses of size i, so C has at most $2\theta_{i-1}$ supersets of size i and the lemma follows. □

□

Lemma 19. *The total number of clauses of size $\leq i$, that are ever added along any path is at most $\beta_i n$.*

Proof. We prove this by induction on i. For $i = 1$, there are at most n literals added along any path and $\beta_1 = 1$. Assume at most β_{i-1} clauses of size $\leq i-1$ are added along a path. Each added clause of size i is either present in the final formula at the leaf or gets eliminated by a smaller clause somewhere on the path. As observed before, there are at most $\theta_{i-1}n$ clauses of size i in the formula at a leaf. By induction hypothesis and lemma 18, each of at most $\beta_{i-1}n$ of the added smaller clauses can eliminate at most $2\theta_{i-1}$ clauses of size i. Thus total number of clauses of size $\leq i$ added along the path is $\leq \theta_{i-1}n + 2\beta_{i-1}\theta_{i-1}n \leq 4\beta_{i-1}\theta_{i-1}n = \beta_i n$. □

□

Lemma 20. *Along any path, there are at most kn/α petal-additions.*

Proof. By lemma 19, there are at most $\beta_i n$ clauses of size i that are ever added along the path. Since each petal addition with petal size i adds at least θ_i petals, there are at most $\beta_i n/\theta_i$ petal additions with petal size i. Summing over i gives the result. □

□

Now we can easily prove that the number of leaves is at most $2^{\epsilon n}$ completing the proof of Theorem 15. As mentioned before, every leaf is uniquely determined by a sequence of center/petal additions. By lemma 19 and 20, the length of any path is at most $\beta_{k-1}n$ and the number of petal additions is at most kn/α. Thus the number of leaves is at most

$$\sum_{r=0}^{kn/\alpha} \binom{\beta_{k-1}n}{r} \leq 2^{\epsilon n}$$

We note that $kn/\alpha \ll \beta_{k-1}n$ and we can use the approximation $\sum_{r=0}^{\delta l} \binom{l}{r} \approx 2^{H(\delta)l}$ for small δ where $H(\delta)$ is the entropy function $H(\delta) = \delta \log(1/\delta) + (1-\delta)\log(1/(1-\delta))$. □

As observed before, Theorem 15 implies that the problems k-SAT, k-SAT(c_k), INDEPENDENT SET, 3-Colorability are all SERF-equivalent to each other. If any of these problmes has a $2^{o(n)}$ time algorithm, then so do the others.

LECTURE 3
Toda's Theorem: $PH \subseteq P^{\#P}$

In this talk, we introduce the counting class $\#P$ and prove Toda's theorem, that the polynomial hierarchy is contained in $P^{\#P}$.

Theorem 21. *([**Toda91**]) $PH \subseteq P^{\#P}$*

1. Classes $\#P$ and $\oplus P$

Let us first see examples of problems in $\#P$.

(1) $\#SAT$ **:** Given a boolean formula ϕ, what is the number of satisfying assignments of ϕ ?
(2) $\#PATH$ **:** Given a directed graph G with two specific vertices s and t, what is the number of simple paths from s to t ?
(3) Network Reliability : Given a graph G, what is the probability that G become disconnected if every edge in G fails with probability p ?
(4) Permanent : Given an $n \times n$ integer matrix $A = (a_{ij})$, compute its permanent.

$$Per(A) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} a_{i\sigma(i)}$$

where $S(n)$ is the set of all permutations on n-set.

We formally define the class $\#P$ as follows :

Definition 22. *A function $f : \{0,1\}^* \to \mathbf{N}$ is in $\#P$ if there exists a polynomial time NTM M such that $\forall x \in \{0,1\}^*$, $f(x)$ equals the number of accepting paths of M on input x.*

We note that counting the number of assignments to a SAT formula is at least as hard as deciding whether the formula is satisfiable (or a tautology), so $NP \subseteq P^{\#P}$ (and $coNP \subseteq P^{\#P}$).

Definition 23. *A function f is $\#P$-complete if $f \in \#P$ and $\forall g \in \#P$, $g \in P^f$.*

Since the Cook-Levin reduction from an NP language L to a SAT formula maintains a one to one correspondence between the accepting paths of NTM deciding L to the satisfying assignments to the corresponding SAT formula, it is clear

that $\#SAT$ is complete for $\#P$. In fact, Valiant [**Val79**] shows that all the problems mentioned above are $\#P$-complete. Note that the s-t connectivity problem on directed graphs is in NL, but the problem $\#PATH$ is $\#P$-complete!

Toda's Theorem came as a surprise to most people in 1989, since it was widely believed that alternation (the feature of the polynomial hierarchy) could not have anything do with the ability of count the number of solutions (a feature of $\#P$).

We now define a class $\oplus P$ which we use as an intermediate class in the proof of Theorem 21.

Definition 24. *A language $L \in \oplus P$ iff there exists a polynomial time NTM M such that $x \in L$ iff the number of accepting paths of M on input x is odd.*

Definition 25.

$\oplus SAT = \{\phi \mid \phi$ *is a boolean formula with an odd number of satisfying assignments* $\}$.

A simple modification of the Cook-Levin theorem shows that $\oplus SAT$ is complete for $\oplus P$.

Now define a randomized reduction between two languages.

Definition 26. *Language A reduces to language B under a randomized reduction, denoted $A \leq_r B$, if there exists a deterministic, polynomial time computable function $f : \{0,1\}^* \to \{0,1\}^*$ and a polynomial $p()$ such that*

$$\forall x \in A, \quad Pr_{y \in \{0,1\}^{p(|x|)}}[f(x,y) \in B] \geq 2/3$$

$$\forall x \notin A, \quad Pr_{y \in \{0,1\}^{p(|x|)}}[f(x,y) \in B] \leq 1/3$$

We prove Theorem 21 in two steps. First we show that any $L \in PH$ can be randomly reduced to $\oplus SAT$ and then we show that this reduction can be transformed into a (deterministic) reduction to a $\#P$ problem.

Throughout, we will use characterization of the classes $\#P$ and $\oplus P$ in terms of uniform circuit families, which is left as an exercise.

Definition 27. *Let $\{\mathcal{C}_n\}_{n \geq 1}$ be a circuit family of size $S(n)$ for input of length n. We say that this is a Direct Connect uniform (DC uniform) family if all the following functions can be computed in deterministic $O(log(S(n)))$ time :*

- *$TYPE(n,i)$ = the type (AND, OR, NOT, INPUT, OUTPUT) of gate i in circuit $\mathcal{C}_n$.*
- *$IN(n,i,j) = k$ where k is j^{th} in the ordered list of indices to the gates that feed into gate i, $k = NONE$ if there is no j^{th} gate.*
- *$OUT(n,i,j) = k$ where k is j^{th} in the ordered list of indices to the gates that gate i feeds into, $k = NONE$ if there is no j^{th} gate.*

Theorem 28. *$L \in PH$ iff L can be computed by a DC uniform circuit family $\{\mathcal{C}_n\}$ that*

- *uses AND, OR, NOT gates.*
- *has size $2^{n^{O(1)}}$ and constant depth.*
- *gates can have unbounded (exponential) fanin.*
- *the NOT gates appear only at the input level.*

Theorem 29. *$L \in \oplus P$ iff L can be computed by a DC uniform circuit family $\{\mathcal{D}_n\}$ that*

- *uses AND, OR, NOT, XOR gates.*

- *has size* $2^{n^{O(1)}}$ *and constant depth.*
- *XOR gates can have unbouned (exponential) fanin, but the AND, OR gates have fanin at most* $n^{O(1)}$.
- *NOT gates can appear anywhere in the circuit.*

Note that the circuits have exponential size, but they have a succinct representation in terms of a TM which can systematically generate any required portion of the circuit.

2. The Main Lemma

In this section we prove our main lemma that every language in polynomial hierarchy is randomly reducible to $\oplus SAT$.

Lemma 30. $\forall L \in PH, \quad L \leq_r \oplus SAT.$

Proof. Theorem 28 gives a characterization of L in terms of DC uniform circuit family $\{\mathcal{C}_n\}$. We use a randomized reduction to transform $\{\mathcal{C}_n\}$ into another DC uniform family $\{\mathcal{D}_n\}$ which uses $\oplus$ gates with exponential fanin and AND, OR gates with fanin $n^{O(1)}$. Theorem 29 then shows that the language computed by $\{\mathcal{D}_n\}$ can be expressed as a $\oplus SAT$ problem (since $\oplus SAT$ is complete for $\oplus P$).

The reduction uses the Valiant-Vazirani lemma [**VV86**].

Definition 31. *If* $S \subseteq \{0,1\}^N = GF(2)^N$, *and* $a_1, \ldots, a_k \in GF(2)^N$, *then*

$$S \bigcap (a_1, \ldots, a_k)^{\perp} = \{s \in S : \quad s \cdot a_1 = s \cdot a_2 = \ldots = s \cdot a_k = 0\}.$$

Lemma 32. *(* [**VV86**]*) If* S *is nonempty and* $i \in \{0, 1, \ldots, N\}$ *and* $a_1, \ldots, a_i \in GF(2)^N$ *are chosen uniformly at random then*

$$Pr[\ |S \bigcap (a_1, \ldots, a_i)^{\perp}| \ \textit{is odd}] \geq \frac{1}{4N}.$$

Let the size of circuit $\mathcal{C}_n$ be 2^N where $N = n^{O(1)}$. Consider conversion of an OR gate g into XOR gate (Figure 1). The inputs $w_1, w_2, \ldots$ to the gate g can be indexed by strings in $GF(2)^N$. We choose i randomly from $\{0, 1, \ldots, N\}$ and strings $a_1, \ldots, a_i$ randomly from $GF(2)^N$. Then we replace the gate g by $\oplus$ gate with the same number of inputs, but the input w_s is replced by logical AND of w_s and a predicate p_s which is 1 iff $s \cdot a_1 = \ldots = s \cdot a_i = 0$. We note that if output of gate g is 0, so is output of $\oplus$ gate. If output of gate g is 1, there is a nonempty set S of strings s such that $w_s = 1$. With probability $\geq 1/4N$, the number of strings $s \in S$ such that $s \in (a_1, \ldots, a_i)^{\perp}$ is odd and the output of $\oplus$ gate is 1. Thus the $\oplus$ gate is equivalent to gate g with probability at least $1/4N$. We can amplify this probability by taking OR of $8N^2$ independent copies of this $\oplus$ gate. The resulting circuit g_1 satisfies,

$$Pr[g_1(w_1, w_2 \ldots.) \neq g(w_1, w_2, \ldots)] \leq (1 - 1/4N)^{8N^2} \leq e^{-2N}.$$

We apply this transformation to every gate in circuit $\mathcal{C}_n$. (The AND gates are transformed using De-Morgan's law). We use the same random choice of $i, a_1, \ldots, a_i$ for each gate, thus using only $n^{O(1)}$ random bits. The new circuit $\mathcal{D}_n$ depends on the choice of the random bits. It agrees with $\mathcal{C}_n$ on all inputs of lengh n with probability $\geq 1 - 2^n 2^N e^{-2N} \geq 2/3$. It is clear that $\{\mathcal{D}\}_n$ is a DC uniform family satisfying conditions of Theorem 29. As observed before, we can construct a $\oplus SAT$ question that decides whether an input is accepted by circuit $\mathcal{D}_n$. □

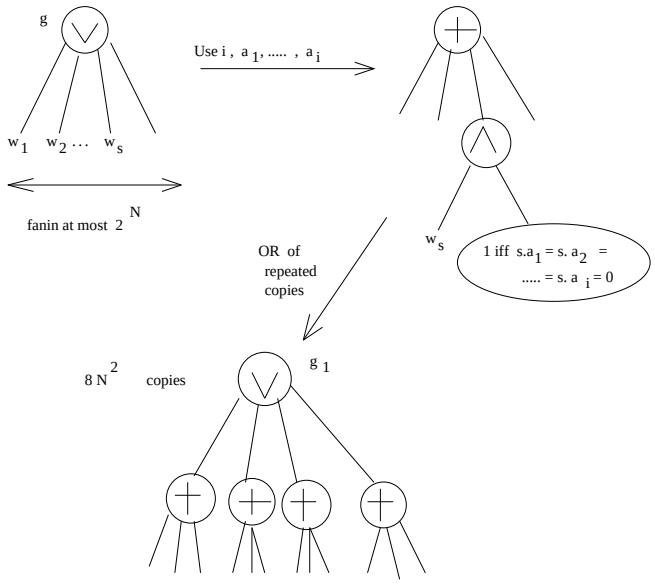

Figure 1. Probabilistic conversion of OR gate to $\oplus$ gate

3. Proof of Theorem 21

For $L \in PH$, lemma 30 gives a reduction $x \to \phi_r$ from input x to a SAT formula ϕ_r using a random string r of length m such that

$$x \in L \Longrightarrow Pr_r[\phi_r \in \oplus SAT] \geq 2/3$$

$$x \notin L \Longrightarrow Pr_r[\phi_r \in \oplus SAT] \leq 1/3$$

We will construct a $\#P$ question that counts the number of random strings r such that $\phi_r \in \oplus SAT$. Using a $\#P$-oracle, we then know the probability $Pr_r[\phi_r \in \oplus SAT]$ and we can decide whether $x \in L$, proving Toda's theorem.

We define a transformation of ϕ_r to another SAT formula ϕ_r^* such that

$$\phi_r \in \oplus SAT \Longrightarrow \#(\phi_r^*) \equiv -1 (mod\ 2^{m+1})$$

$$\phi_r \notin \oplus SAT \Longrightarrow \#(\phi_r^*) \equiv 0\ (mod\ 2^{m+1}).$$

Here $\#(\psi)$ is the number of satisfying assignments to formula ψ.

Now we construct a NP machine (Figure 2) that guesses a random string r, computes ϕ_r and ϕ_r^*, branches out according to an assignment for ϕ_r^* and then checks whether this assignment satisfies ϕ_r^*. Clearly,

$$\#accepting\ paths = \sum_r \#(\phi_r^*) \tag{2}$$

$$\equiv -|\{r|\phi_r \in \oplus SAT\}|\ (mod\ 2^{m+1}) \tag{3}$$

Since there are only 2^m random strings r, the number of accepting paths reduced modulo 2^{m+1} gives the number of random strings r such that $\phi_r \in \oplus SAT$. By definition of $\#P$, a $\#P$-oracle can tell the number of accepting paths of this NTM on given input and the proof is complete.

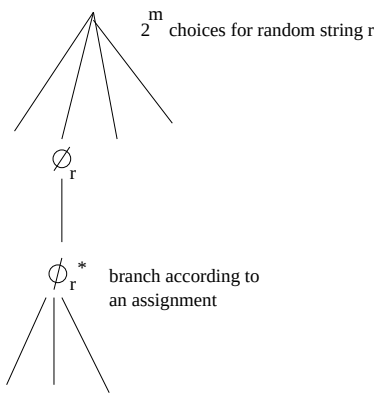

Figure 2

We now give the transformation $\phi \to \phi^*$. For formulas $\varphi(x_1, \ldots, x_n)$ and $\psi(y_1, \ldots, y_m)$, let $\varphi + \psi$ and $\varphi \cdot \psi$ be formulas such that

$$\#(\varphi + \psi) = \#(\varphi) + \#(\psi)$$
$$\#(\varphi \cdot \psi) = \#(\varphi) \cdot \#(\psi)$$

Such formulas can be constructed easily with only constant factor increase in size. One can easily check that

$$\#(\psi) \equiv -1 (mod\ 2^{2^i}) \Longrightarrow \#(4\psi^3 + 3\psi^4) \equiv -1 (mod\ 2^{2^{i+1}}) \text{ and} \tag{4}$$

$$\#(\psi) \equiv 0\ (mod\ 2^{2^i}) \Longrightarrow \#(4\psi^3 + 3\psi^4) \equiv 0\ (mod\ 2^{2^{i+1}}). \tag{5}$$

Let $\phi_0 = \phi$ and $\phi_{i+1} = 4\phi_i^3 + 3\phi_i^4$. Let

$$\phi^* = \phi_{\lceil \log(m+1) \rceil}$$

Repeated use of equations 4, 5 shows that if $\#(\phi)$ is odd, $\#(\phi^*) \equiv -1\ (mod\ 2^{m+1})$ and if $\#(\phi)$ is even, $\#(\phi^*) \equiv 0\ (mod\ 2^{m+1})$. Also, size of ϕ^* is only polynomially larger than size of ϕ. □

4. Open Problems

- What is the exact power of $\oplus SAT$ and $\#SAT$?
- What is the complexity of computing approximation to permanent?
- What is the average case complexity of $n \times n$ permanent modulo small prime, say 3 or 5 ? Note that for a prime $p > n$, random self reducibility of permanent implies that if permanent is hard to compute on at least one input then it is hard to compute on $1 - O(p/n)$ fraction of inputs, i.e. hard to compute on average [**Lip89**].

BIBLIOGRAPHY

[ALMSS92] S. Arora, C. Lund, R. Motawani, M. Sudan, M. Szegedy. *Proof verification and hardness of approximation problems.* In Proceedings of the 33rd IEEE FOCS, 14-23, 1992.

[AS92] S. Arora, S. Safra. *Probabilistic checking of proofs : A new characterization of NP.* In Proceedings of the 33rd IEEE FOCS, 2-13,1992.

[Bab85] L. Babai. *Trading group theory for randomness.* In Proceedings of the 17th ACM STOC, 421-429, 1985.

[BFL91] L. Babai, L. Fortnow, C. Lund. *Non-deterministic exponential time has two-prover interactive protocols.* Computational Complexity, 113-121, 1988.

[BLR90] M. Blum, M. Luby, R. Rubinfeld. *Self-testing/correcting with applications to numerical problems.* In Proceedings of the 22nd ACM STOC, 73-83, 1990.

[BK89] M. Blum, S. Kannan. *Designing programs that check their work.* In Proceedings of the 21st ACM STOC, 86-97, 1989.

[DF95] R.G. Downey, M.R. Fellows. *Fixed parameter tractability and completeness I : Basic theory.* SIAM J on Computing, 24, 873-921, 1995.

[FGLSS91] U. Feige, S. Goldwasser, L. Lovász, S. Safra, M. Szegedy. *Approximatin clique is almost NP-complete.* In Proceedings of the 32nd IEEE FOCS, 2-12, 1991.

[GMR89] S. Goldwasser, S. MIcali, C. Rackoff. *The knowledge complexity of interactive proofs.* SIAM J. on Computing, 18, 186-208, 1989.

[Hå96] J. Håstad. *Clique is hard to approximate within* $n^{1-\epsilon}$. In Proceedings of the 37th IEEE FOCS, 627-636, 1996.

[Hå97] J. Håstad. *Some optimal inapproximability results.* In Proceedings of the 29th ACM STOC, 1-10, 1997.

[IPZ98] R. Impagliazzo, R. Paturi, F. Zane. *Which problems have strongly exponential complexity ?* In Proceedings of the 39th IEEE FOCS, 653-662, 1998.

[Joh74] D.S. Johnson. *Approximation algorithms for combinatorial problems.* Journal of Computer and Systems Sciences, 9:256-278, 1974.

[KZ97] H. Karloff and U. Zwick. A 7/8 approximation for MAX-3SAT? In *Proceedings of the 38th IEEE FOCS*, 406–415, 1997. (Latest version available from authors has stronger results.)

[Lev] L. Levin. Average case complete problems. *SIAM Journal on Computing*, 15:285–286, 1986.

[Lip89] R. Lipton. *Efficient checking of computations.* In Proceedings of 6th STACS, 1989.

[LFKN92] C. Lund, L. Fortnow, H. Karloff, N. Nisan. *Algebraic methods for interactive proof systems.* Journal of the ACM, 39(4), 859-868, 1992.

[LPS86] A. Lubotzky, R. Phillips, P. Sarnak. *Explicit expanders and the Ramanujan conjectures.* In Proceedings of the 18th ACM STOC, 1986.

[Pap94] C. H. Papadimitriou. On the Complexity of the Parity Argument and Other Inefficient Proofs of Existence. *JCSS* **48**(3):498–532, 1994.

[Pap94b] C. H. Papadimitriou. Computational Complexity. Addison Wesley, 1994.

[SV97] A. Sahai, S. Vadhan. *A complete promise problem for statistical zero-knowledge.* In Proceedings of the 38th IEEE FOCS, 1997.

[Sha92] A. Shamir. *IP=PSPACE.* Journal of the ACM, 39(4), 869-877, 1992.

[Toda91] S. Toda. *PP is as hard as the polynomial hierarchy.* SIAM J. Computing, 865-877.

[Val79] L. Valiant. *The complexity of enumeration and reliability problems.* SIAM J. Computing, 8, 410-421.

[VV86] L. Valiant, V. Vazirani. *NP is as easy as detecting unique solutions.* Theoretical Computer Sciecne, 47, 85-93.