A MACHINE MODEL FOR THE COMPLEXITY OF NP-APPROXIMATION PROBLEMS

RICHARD CHANG[†]
UMBC

This paper investigates a machine-based model for the complexity of approximating the CLIQUE problem. The model consists of nondeterministic polynomial time Turing machines with limited access to an NP-complete oracle. Approximating the CLIQUE problem is complete for classes of functions computed by such machines.

1 Introduction

Many introductions to the theory of NP-completeness make some mention of approximating NP-complete problems. The usual story line says that even though solving an NP-complete problem exactly may be intractable, it is sometimes possible to find an approximate solution in polynomial time. The intuitive assumption is that finding an approximate solution should be easier than finding the exact solution. While this assumption holds for a long list of approximable problems (Bin Packing, Euclidean Travelling Salesman Problem, Set Cover, Subset Sum, etc.), recent works have shown that for the CLIQUE problem, even coarse approximations are not achievable in polynomial time unless P = NP.[1,2,3] Furthermore, these non-approximability results extend to other NP-complete problems; most notably to Graph Coloring, Set Cover, MAX3SAT and all MAXSNP-complete problems.[3,4] Nevertheless, showing that an NP-approximation problem cannot be solved in polynomial time only shows that the problem is difficult to compute; *it does not show that solving the approximation problem is just as hard as finding the exact solution.* Thus, it remains possible that approximating CLIQUE is in fact easier than finding the exact solution to CLIQUE.

In this paper, we point out that for several NP-optimization problems, we can *prove*, not just intuit, that finding the exact solution is harder than finding an approximate solution, under the assumption that the Polynomial Hierarchy (PH) does not collapse. To keep the exposition simple, we focus on determining the precise complexity of finding approximations to the CLIQUE problem, but the results extend to many other NP-approximation problems.

[†]Department of Computer Science and Electrical Engineering, University of Maryland Baltimore County, 1000 Hilltop Circle, Baltimore, MD 21250, USA. Email: `chang@umbc.edu` Supported in part by National Science Foundation research grants CCR-9309137 and CCR-9610457 and by the University of Maryland Institute for Advanced Computer Studies.

Our first question is:

Question 1: Is finding the vertices of the largest clique more difficult than merely finding the vertices of a 2-approximate clique (that is, a clique with at least half the size of the largest clique)?

Answer: You cannot reduce finding the largest clique to finding a 2-approximate clique, unless PH collapses.

Three recent studies have addressed this question in one form or another. The first approach, by Chang, Gasarch and Lund, obtains upper and lower bounds on the number of queries to a SAT oracle needed to approximate the *size* of the maximum clique.[5,6,7] Specifically, approximating the clique size within a factor of 2 is complete for $\mathrm{PF}^{\mathrm{SAT}[\log\log n+O(1)]}$, the class of polynomial-time Turing machines which ask at most $\log\log n + O(1)$ queries to SAT. A consequence of this result is that finding the size of the maximum clique cannot be reduced to approximating the size, unless P = NP. However, these results deal with the complexity of approximating the *size* of the largest clique and not on the complexity of finding the vertices of an approximate clique. Thus, these results do not resolve Question 1.

A second set of results does consider the difficulty of finding the *solutions* to NP-approximation problems.[8] Khanna, Motwani, Sudan and Vazirani showed that every problem in APX can be reduced to MAX3SAT using an approximation preserving reduction. APX is the set of NP-optimization problems which have constant-factor polynomial-time approximation algorithms. Similarly, they showed that every NP-optimization problem which has a polynomial-factor polynomial-time approximation algorithm can be reduced to CLIQUE. These results do not address Question 1 directly, because they are more useful for comparing the complexities of different NP-optimization problems than for comparing the complexities of a single NP-optimization problem with different approximation factors. However, they do provide us with some very robust gap-creating reductions which we will need later.

In a third approach, Crescenzi, Kann, Silvestri and Trevisan also considered problems complete for APX under various types of approximation preserving reductions.[9,10] However, they took a novel direction and measured the complexity of NP-approximation problems by looking at which languages can be recognized by polynomial-time machines which have a function oracle that solves the approximation problem. Then, using a classic census argument, they were able to show that finding the maximum clique cannot be reduced to finding an approximate clique unless PH collapses. Thus, Question 1 is finally resolved. In Section 3, we will review these results in greater detail.

The answer to Question 1 tells us that our intuition is indeed correct—finding an exact solution to CLIQUE is harder than merely approximating CLIQUE. However, this answer only tells us the relative complexity of approximating CLIQUE. It does not provide us with a framework for comparing the complexity of approximating CLIQUE with respect to the complexity of other problems (especially with respect to problems that are not optimization problems). Hence, we ask ourselves a second question:

Question 2: What computational resources can be used to find the vertices of a 2-approximate clique?

Answer: Nondeterminism and $\log \log n + O(1)$ queries to SAT.

It turns out that by combining the techniques used in the three studies described above, we can show that an $\mathrm{NPF}_b^{\mathrm{SAT}[\log\log n+O(1)]}$ machine (that is, an NP machine which asks at most $\log \log n + O(1)$ queries to SAT *in the entire computation tree*) can compute a multivalued function that outputs the vertices of a 2-approximate clique. Moreover, every multivalued function computed by an $\mathrm{NPF}_b^{\mathrm{SAT}[\log\log n+O(1)]}$ machine can be reduced to approximating CLIQUE within a factor of 2. Thus, we obtain a nice contrast between the complexity of finding the size of an approximate clique versus the complexity of finding the vertices of an approximate clique. Namely:

- Finding a number x such that the size of the largest clique is between x and $2x$ is complete for $\mathrm{PF}^{\mathrm{SAT}[\log\log n+O(1)]}$.

- Finding the vertices of a clique X such that the size of the largest clique is less than $2|X|$ is complete for $\mathrm{NPF}_b^{\mathrm{SAT}[\log\log n+O(1)]}$.

Thus, the difference between approximating the *clique size* and approximating the clique itself is just nondeterminism. These results can be extended to other approximation factors (e.g., constants, $\log n$, $n^{1/a}$, etc.) and to other NP-approximation problems. In general, closer approximations can be obtained using more queries.

However, this result by itself does not show that approximating CLIQUE within a factor of 2 *requires* $\log \log n$ queries or that we need to use a nondeterministic Turing machine. So, we arrive at our third question:

Question 3: Could fewer resources be used to find a 2-approximate clique?

Answer: You need to have nondeterminism, unless P = NP, and you need at least $\log \log n - c$ queries for some constant c, unless PH collapses.

In the deterministic case, we know that for all $f(n) \in O(\log n)$

$$\mathrm{PF}^{\mathrm{SAT}[f(n)]} \subsetneq \mathrm{PF}^{\mathrm{SAT}[f(n)+1]} \text{ unless } \mathrm{P} = \mathrm{NP}.$$

For technical reasons, the methods used to prove these results do not extend to nondeterministic machines. However, we can prove similar results by relying on the substantial body of research on the Boolean Hierarchy. It turns out that for $f(n) \in O(\log n)$, $\mathrm{NP}_b^{\mathrm{SAT}[f(n)]}$, the set of *languages* accepted by an NP machine making $f(n)$ queries to SAT, is exactly the $2^{f(n)+1} - 1$ level of the Boolean Hierarchy. Furthermore,

$$\mathrm{NPF}_b^{\mathrm{SAT}[f(n)]} = \mathrm{NPF}_b^{\mathrm{SAT}[f(n)+1]} \Longrightarrow \mathrm{NP}_b^{\mathrm{SAT}[f(n)]} = \mathrm{NP}_b^{\mathrm{SAT}[f(n)+1]}.$$

Thus, using the techniques of Chang, Gasarch and Lund, we can show that if CLIQUE can be approximated within a factor of 2 using only $\log\log n - c$ queries, then $\mathrm{NPF}_b^{\mathrm{SAT}[\log\log n - c]} = \mathrm{NPF}_b^{\mathrm{SAT}[\log\log n - c+1]}$. Therefore, the Boolean Hierarchy collapses which in turn implies that PH collapses. This result is a refinement of the proof by Crescenzi, Kann, Silvestri and Trevisan and yields a slightly deeper collapse of PH.

It is interesting to see that the Boolean Hierarchy, which was studied mostly for its mathematical elegance, is actually needed to resolve natural questions about the complexity of approximation problems. In fact much of the intricacy of the Boolean Hierarchy can now be interpreted in this new framework. For example, we know that a deeper collapse of the Boolean Hierarchy results in a deeper collapse of PH.[11,12] Interpreted as an approximation result, these theorems say that a reduction from CLIQUE to a coarser approximation of CLIQUE results in a deeper collapse of PH. Similarly, theorems about the behavior of the complete languages in the Boolean Hierarchy under randomized reductions can now be used to investigate the possibility that CLIQUE can be *randomly* reduced to approximating CLIQUE.[13,14] While this is quite exciting for the few people who are still investigating the Boolean Hierarchy, it is somewhat lamentable to note that some of the theorems that we need now have never been properly written down because they were considered too esoteric even to the people who worked with the Boolean Hierarchy.

In the rest of the paper, we give a more detailed description of the results we have mentioned. Where appropriate we will provide a sketch of the proof, but more importantly we need to provide exact definitions of the terminology we have been using. In particular, we have to define what we mean by "reduction" in each situation.

2 Definitions and Notation

An NP-optimization problem is usually defined in four parts:

1. a polynomial-time decidable set of instances of the problem (e.g., undirected graphs).

2. a polynomial-time decidable set of solutions for each instance where each solution has length polynomial in the size of the instance (e.g., subgraphs of the graph that are cliques).

3. a polynomial-time computable objective function (e.g., the size of the subgraph).

4. whether the problem is a maximization or minimization problem.

Some examples of NP-optimization problems that we will work with are CLIQUE, Graph Coloring and MAX3SAT. In this paper, we have to pay careful attention to the difference between computing the *size* of the maximum clique of a graph G, which we denote as $\omega(G)$, and *finding the vertices* of a maximum clique, which we denote as $\textsc{MaxClique}(G)$. Now we can distinguish between approximating $\omega(G)$ and approximating $\textsc{MaxClique}(G)$.

Definition 1 Let G be an undirected graph with n vertices. We say that a number x $k(n)$-approximates $\omega(G)$ if $x \leq \omega(G) < k(n)x$. We also say that a function f $k(n)$-approximates ω if $f(G)$ $k(n)$-approximates $\omega(G)$ for all graphs G.

Definition 2 Let G be an undirected graph with n vertices. We say that a subgraph X of G $k(n)$-approximates $\textsc{MaxClique}(G)$ if X is a clique and $|X|$ $k(n)$-approximates $\omega(G)$. We also say that a multivalued function f $k(n)$-approximates $\textsc{MaxClique}$ if every output of $f(G)$ is a subgraph X of G such that X $k(n)$-approximates $\textsc{MaxClique}(G)$.

Similarly, we use $\chi(G)$ to denote the chromatic number of a graph and $\textsc{Color}(G)$ to denote an assignment of colors to each node of a graph that uses $\chi(G)$ colors. We also use $\textsc{MaxSat}(F)$ to denote the largest number of simultaneously satisfiable clauses in a 3CNF Boolean formula F. Then, we can define approximating $\chi(G)$, $\textsc{Color}(G)$ and $\textsc{MaxSat}(F)$ analogously.

Most research on NP-optimization problems has been devoted to classifying the problems according to the degree of approximability in polynomial time. For example, the class of NP-optimization problems which have constant-factor polynomial-time approximation algorithms is usually called

APX and the class of NP-optimization problems that have polynomial-factor polynomial-time approximation algorithms is called poly-APX.

Here, we look at NP-optimization problems in a different light. Instead of asking for the degree of approximability in polynomial time, we fix the approximation factor and ask for the least complex function that can approximate the NP-optimization problem within that factor.[a] This approach is much closer to the familiar resource-bounded complexity measures such as time and space.[15,16] This approach has not been used before because researchers have not been able to define a machine model for NP-approximation problems whereby restricting the resources of the machine results in a restriction of the degree of approximability achievable by the machine. The lack of a good machine model was pointed out by Papadimitriou and Yannakakis when they defined MAXSNP.[17] They lamented that "computation is an inherently unstable, non-robust mathematical object, in the sense that it can be turned from non-accepting to accepting by changes that would be insignificant in any metric." Well, we have just such a robust machine model now. They are the bounded query machines which we define below:

Definition 3 Let $\mathrm{PF}^{X[f(n)]}$ denote the set of functions computable by polynomial-time Turing machines which ask at most $f(n)$ queries to the oracle X. For notational convenience n is the size of the input which might not be the length of the encoding of the input. In particular, we consider the number of vertices in a graph to be its size.

Definition 4 Let $\mathrm{NPF}_b^{X[f(n)]}$ denote the set of total multivalued functions computable by nondeterministic polynomial-time Turing machines which ask at most $f(n)$ queries to the oracle X in the entire computation tree. Similarly, let $\mathrm{NP}_b^{X[f(n)]}$ be the set of *languages* recognized by such machines.

For the nondeterministic bounded query machines, the restriction that we count queries in the entire computation tree has been studied by Wagner and by Book, Long and Selman in the context of relativization.[18,19] In this paper, we are primarily interested in NP machines with a SAT oracle. In this case, if we allowed every computation path just one query, then the NP machine is as powerful as an NP machine that asks polynomially many queries in each path. Hence, counting queries only makes sense when we look at the number of queries in the entire tree.

Now that we have defined some function classes, we need to define reductions between functions so we can talk about completeness. Again, our

[a]Actually, the term NP-optimization problem is somewhat of a misnomer since in general, an NP machine cannot find the optimal solution to an NP-optimization problem.

definition is a departure from the usual approximation preserving reductions, because we want to consider reductions between all functions in the bounded query classes, some of which might not be approximation problems.

Definition 5 Let f and g be two functions. We say that f many-one reduces to g (written $f \leq^{\mathrm{P}}_{\mathrm{mt}} g$) if there exist polynomial-time functions T_1 and T_2 such that $f(x) = T_2(g(T_1(x)), x)$. Intuitively, f reduces to g if the input to f can be transformed into an input to g via T_1, so that the value of g on this input along with the original input allows T_2 to compute f. These reductions are also known as metric reductions.

It would be most convenient for us to use the terms "$k(n)$-approximating ω" and "$k(n)$-approximating MaxClique" to represent functions rather than properties of functions. However, these functions must be multivalued because there could be many x's such that $x \leq \omega(G) < k(n)x$ and we do not care which particular x is given by the function. Similarly, approximating MaxClique is also multivalued. Thus, we use "$k(n)$-approximating ω" to refer to the multivalued function which on input G with n vertices outputs all possible values x, such that x $k(n)$-approximates $\omega(G)$. There is a difference between a multivalued function f that $k(n)$-approximates ω and "$k(n)$-approximating ω" because f might produce only some of the values that $k(n)$-approximates $\omega(G)$ (maybe just one such value). The term "$k(n)$-approximating MaxClique" is defined analogously. It may seem awkward to have to consider multivalued functions, but this is really the more natural approach. Note that even computing MaxClique exactly is multivalued because the largest clique in a graph might not be unique.

When we consider reductions between multivalued functions, we cannot use the $\leq^{\mathrm{P}}_{\mathrm{mt}}$-reductions defined above. In this case, we want to say that f reduces to g if *every* output of g helps us compute *some* output of f. We define this reduction formally:

Definition 6 Let f and g be two multivalued functions. We say that f many-one reduces to g (written $f \leq^{\mathrm{P}}_{\mathrm{mv}} g$) if there exist polynomial-time computable functions T_1 and T_2 such that for all x and all possible outputs y of $g(T_1(x))$, $T_2(y, x)$ is a possible output of $f(x)$. This is an extension of metric reductions for single-valued functions.

In this terminology, when a multivalued function f $\leq^{\mathrm{P}}_{\mathrm{mv}}$-reduces to $k(n)$-approximating ω then f $\leq^{\mathrm{P}}_{\mathrm{mv}}$-reduces to *every* multivalued function that $k(n)$-approximates ω. Also, when we say that $k(n)$-approximating ω $\leq^{\mathrm{P}}_{\mathrm{mv}}$-reduces to a multivalued function f then *some* of the functions which $k(n)$-approximates ω $\leq^{\mathrm{P}}_{\mathrm{mv}}$-reduce to f. It is not the case that every function which $k(n)$-approximates ω reduces to f, because there may be some g which

$k(n)$-approximates ω whose values are never output by the T_2 function. This is not an anomaly because there are uncomputable functions which $k(n)$-approximate ω. Similar observations should be made about MaxClique and $k(n)$-approximating MaxClique.

There is another point we should make about reductions between multivalued functions. As we mentioned above, it is possible for f to $\leq^{\mathrm{P}}_{\mathrm{mv}}$-reduce to g via T_1 and T_2, but not all outputs of f are produced by T_2. Thus, in the case where some values of f are easier to compute than other values on the same input, it is possible for f to $\leq^{\mathrm{P}}_{\mathrm{mv}}$-reduce to a function g with much lower complexity. A stronger reduction from f to g will avoid this situation.

Definition 7 Let f and g be two multivalued functions. We say that f *strongly* $\leq^{\mathrm{P}}_{\mathrm{mv}}$-reduces to g if f $\leq^{\mathrm{P}}_{\mathrm{mv}}$-reduces to g via T_1 and T_2 and for all x, if z is an output of f, then there exists an output y of $g(T_1(x))$ such that $T_2(y, x) = z$.

Also, a reduction from MaxClique to 2-approximating MaxClique does not immediately imply a reduction from ω to 2-approximating ω even though, intuitively, the first reduction preserves more structure. The reason is that the T_2 function which computes the solution to MaxClique must be given the *vertices* of a 2-approximate clique and not just the size.

With most of our terminology well-defined, we can finally phrase Question 1 more accurately as:

Question 1:

Does MaxClique $\leq^{\mathrm{P}}_{\mathrm{mv}}$-reduce to 2-approximating MaxClique?

3 Review of Results

3.1 Bounded Query Complexity of Approximations

We will first review a series of papers by Chang, Gasarch and Lund.[5,6,7] These papers consider the complexity of approximating the size of the largest clique in a graph in terms of the number of queries to a SAT oracle. The results in these papers show upper bounds and relative lower bounds on the number of queries needed to compute a function which $k(n)$-approximates ω. These upper and lower bounds differ by a constant $c = 2 + \log 1/\epsilon$. Here ϵ is the constant from the non-approximability result of Arora *et al.* and does not depend on $k(n)$. For example, under the assumption that RP $\neq$ NP, the constant c is less than 6. Some niceness assumptions about $k(n)$ are needed to achieve these results, but they hold for all of the usual approximation factors. For example:[b]

- There exists a function in $\mathrm{PF}^{\mathrm{SAT}[\log\log n]}$ which 2-approximates ω, but no function in $\mathrm{PF}^{X[\log\log n - c]}$ can 2-approximate ω for any oracle X, unless $\mathrm{P} = \mathrm{NP}$.
- There exists a function in $\mathrm{PF}^{\mathrm{SAT}[\log\log n - \log\log\log n]}$ which $\log n$-approximates ω, but no function in $\mathrm{PF}^{X[\log\log n - \log\log\log n - c]}$ can $\log n$-approximate ω for any oracle X, unless $\mathrm{P} = \mathrm{NP}$.
- For constant a, there exists a function in $\mathrm{PF}^{\mathrm{SAT}[\log a]}$ which $n^{1/a}$-approximates ω, but no function in $\mathrm{PF}^{X[\log a - c]}$ can $n^{1/a}$-approximate ω for any oracle X, unless $\mathrm{P} = \mathrm{NP}$.

Recall that Krentel showed that computing $\omega(G)$ exactly requires $\Omega(\log n)$ queries to SAT.[20] Hence, these results show that computing $\omega(G)$ exactly is harder than merely approximating $\omega(G)$. Furthermore, closer approximations require more queries. An extension of these results show that, in fact, approximating $\omega(G)$ is equivalent to queries to SAT in the sense that a closer approximation can be obtained using more oracle queries and also yields answers to more oracle queries. For example:

- Computing ω exactly is $\leq^{\mathrm{P}}_{\mathrm{mt}}$-complete for the class $\mathrm{PF}^{\mathrm{SAT}[O(\log n)]}$.
- Approximating ω within a factor of 2 is $\leq^{\mathrm{P}}_{\mathrm{mt}}$-complete for the class $\mathrm{PF}^{\mathrm{SAT}[\log\log n + O(1)]}$.
- Approximating ω within a factor of $\log n$ is $\leq^{\mathrm{P}}_{\mathrm{mt}}$-complete for the class $\mathrm{PF}^{\mathrm{SAT}[\log\log n - \log\log\log n + O(1)]}$.

We know from results on bounded query function classes that each additional query allows a polynomial-time machine to compute more functions unless $\mathrm{P} = \mathrm{NP}$.[21] Thus, we can conclude that ω does not $\leq^{\mathrm{P}}_{\mathrm{mt}}$-reduce to 2-approximating ω and 2-approximating ω does not $\leq^{\mathrm{P}}_{\mathrm{mt}}$-reduce to $\log n$-approximating ω unless $\mathrm{P} = \mathrm{NP}$.

These results provide a resource-bounded complexity measure for the difficulty of approximating ω. One might even venture to argue that the connection between bounded queries and approximating ω is a natural connection. However, in applications where one would want to find the largest clique, it is not sufficient merely to deliver the size of the largest clique. One would also

[b]Similar results can be proven for Chromatic Number, Set Cover and MAX3SAT, but we concentrate on Clique Size here.

want to know which vertices are in the largest clique. Now, for the purposes of proving the non-approximability of CLIQUE, it is sufficient to show that the size of the largest clique is not approximable, since knowing which vertices are in the clique also provides the size of the clique. However, for our purpose, which is to determine the precise complexity of approximating CLIQUE, it is not at all obvious how the size of a clique relates to the complexity of finding the vertices in the clique. In particular, the results mentioned above do not preclude the possibility that finding the vertices of an approximate clique may be just as hard as finding the vertices of the largest clique.

In general, it is difficult to construct reductions that preserve solutions. For example, the results of mentioned above also show that there is a reduction from 2-approximating the Chromatic Number $\chi(G)$ to 2-approximating ω (since there is a function in $\mathrm{PF}^{\mathrm{SAT}[\log\log n]}$ which 2-approximates χ). It is much more difficult to construct a reduction which produces a 2-approximate graph coloring (a coloring of the entire graph using no more than twice the optimal number of colors) from the vertices of a 2-approximate clique. The obvious approaches would result in an optimal coloring of half the vertices rather than a twice optimal coloring of all the vertices. Fortunately, such reductions are possible using results from Probabilistically Checkable Proofs (PCP).

3.2 Some PCP Magic

The advertised results in the paper by Khanna, Motwani, Sudan and Vazirani show that MAX3SAT is complete for APX under L-reductions with scaling and that CLIQUE is complete for poly-APX under L-reductions with scaling.[8] As a consequence, the closure of MAXSNP under these reductions, $\overline{\text{MAXSNP}}$, is equal to APX. Similarly, for the class RMAX(2), defined by Panconesi and Ranjan, its closure is equal to poly-APX.[22] The significance of these results lies in the fact that MAXSNP and RMAX(2) are defined as syntactic classes rather than computational classes. Hence, we now have a tight connection between approximability and syntax.

As we mentioned before, we are primarily interested in the gap-creating reductions that Khanna *et al.* were able to derive from the recent PCP results rather than the advertised results. They showed that there exists a constant δ, and a polynomial-time function f such that for all 3CNF formulas F with n clauses, $f(F)$ produces a 3CNF formula F' with m clauses, where m depends only on n, such that:

$$\begin{aligned} F \in \mathrm{SAT} &\Longrightarrow \textsc{MaxSat}(F') = m \\ F \notin \mathrm{SAT} &\Longrightarrow \textsc{MaxSat}(F') = (1-\delta)m. \end{aligned}$$

Furthermore, given any assignment of truth values to the variables of F' which satisfies more than $(1-\delta)m$ clauses, a satisfying assignment of F' (where *all* clauses are satisfied) can be found in polynomial time. This in turn produces a satisfying assignment of F. The full assignment can be found in polynomial time because the probabilistically checkable proof of the satisfiability of F uses an error-correcting code which guarantees that any assignment which satisfies most of the clauses has a small Hamming distance from a unique full assignment. The polynomial-time error-correcting algorithm will find this full assignment.

A similar reduction from SAT to CLIQUE can also be obtained. Here we have constants $0 < s < b < d$ and a polynomial-time function f such that for all 3CNF formulas F with n clauses, $f(F)$ produces a graph G with n^d vertices and

$$F \in \text{SAT} \Longrightarrow \omega(G) = n^b$$
$$F \notin \text{SAT} \Longrightarrow \omega(G) = n^s.$$

In this case, if we are given a clique in G with $n^s + 1$ vertices, then we can construct a clique with n^b vertices in polynomial time. This is possible because the construction of Feige *et al.* and the properties of expander graphs guarantee that the vertices of an $n^s + 1$ clique represent an assignment which satisfies at least $(1-\delta)m$ clauses of F' in the previous reduction.[1] The dispersal property of expanders is needed here to guarantee that we have $(1-\delta)m$ *distinct* clauses. The conversion to a fully satisfying assignment of F' in turn produces the clique with n^b vertices. As we shall see in the next section, this reduction will allow us to construct a $\leq^{\mathrm{P}}_{\mathrm{mv}}$-reduction from 2-approximating COLOR to 2-approximating MAXCLIQUE.

Now, given the fact that we can take a relatively small clique and expand it into the largest clique in the graph, we might be tempted to think that there is a reduction from MAXCLIQUE to 2-approximating MAXCLIQUE. However, the reduction above produces a large clique only when $F \in \text{SAT}$. Hence, the existence of a large clique in G only gives us the answer to 1 query to SAT, which is not enough to compute MAXCLIQUE.

3.3 That Census Trick Again

To answer Question 1, we need to use the techniques of Crescenzi, Kann, Silvestri and Trevisan.[9,10] They showed that Maximum Polynomially Bounded Weighted Satisfiability (MPBWS) is complete for APX under PTAS reductions. Again, we won't use this main result here. Instead we are most interested in their use of bounded query languages to measure the complexity of

NP-approximation problems.

So, suppose that MaxClique does $\leq^{\mathrm{P}}_{\mathrm{mv}}$-reduce to 2-approximating MaxClique. For now, let 2MC represent the multivalued function "2-approximating MaxClique", that is, the function which on input G outputs all possible X's such that X 2-approximates MaxClique(G). Consider $\mathrm{P}^{\textsc{MaxClique}[1]}$, the class of languages recognized by polynomial-time machines which asks one query string G and receives a string as a reply. The reply string is one of the possible outputs of MaxClique(G). Since MaxClique is a multivalued function, the machine must accept correctly for all possible values of MaxClique(G). The class $\mathrm{P}^{\mathrm{2MC}[1]}$ is defined similarly.

By assumption MaxClique $\leq^{\mathrm{P}}_{\mathrm{mv}}$ 2MC, so

$$\mathrm{P}^{\textsc{MaxClique}[1]} \subseteq \mathrm{P}^{\mathrm{2MC}[1]}.$$

Krentel showed that computing $\omega(G)$ is $\leq^{\mathrm{P}}_{\mathrm{mt}}$-complete for $\mathrm{PF}^{\mathrm{SAT}[O(\log n)]}$. Thus, it follows that $\mathrm{P}^{\mathrm{SAT}[\log n]} \subseteq \mathrm{P}^{\textsc{MaxClique}[1]}$. We will show below that

$$\mathrm{P}^{\mathrm{2MC}[1]} \subseteq \mathrm{P}^{\mathrm{SAT}[\log\log n+1]}.$$

Therefore, we have

$$\mathrm{P}^{\mathrm{SAT}[\log n]} \subseteq \mathrm{P}^{\textsc{MaxClique}[1]} \subseteq \mathrm{P}^{\mathrm{2MC}[1]} \subseteq \mathrm{P}^{\mathrm{SAT}[\log\log n+1]},$$

which collapses the (extended) Boolean Hierarchy which in turn collapses PH.[18,23]

Now, it is not obvious that $\mathrm{P}^{\mathrm{2MC}[1]} \subseteq \mathrm{P}^{\mathrm{SAT}[\log\log n+1]}$, because the computation of the $\mathrm{P}^{\mathrm{2MC}[1]}$ machine depends on the value of 2MC(G) for some query string G and there is no obvious way for a $\mathrm{P}^{\mathrm{SAT}[\log\log n+1]}$ machine to compute 2MC. However, we can find a number z that 2-approximates $\omega(G)$ using $\log\log n$ queries to SAT, then armed with this information ask one more NP question:

> Assuming that z 2-approximates $\omega(G)$, does the $\mathrm{P}^{\mathrm{2MC}[1]}$ machine accept the input string?

This is an NP question because using z, an NP machine can guess a subgraph X of G with z vertices, check that X is a clique, and simulate the $\mathrm{P}^{\mathrm{2MC}[1]}$ computation using X as the answer to the oracle query. Thus, $\mathrm{P}^{\mathrm{2MC}[1]} \subseteq \mathrm{P}^{\mathrm{SAT}[\log\log n+1]}$.

The proof outlined above uses the same "census trick" that Hemachandra used to show that $\mathrm{P}^{\mathrm{SAT}\|} = \mathrm{P}^{\mathrm{SAT}[\log n]}$.[24] It is also related to Mahaney's original proof that the existence of a sparse NP-complete set collapses PH.[25] In both cases, the theorems can be tightened up using more sophisticated counting techniques and we can expect the same in this situation.[26,27] In the next section, we show how to refine the proof by Crescenzi *et al.* and introduce a new model so we can avoid having to talk about multivalued functions as oracles.

4 A Machine Model for NP-Approximation

The results we reviewed so far describe the complexity of MaxClique in relation to the complexity of approximating MaxClique, but they do not tell us what computational resources are needed to solve these problems. In this section, we develop a machine model that gives us the precise complexity of computing MaxClique and approximating MaxClique. From these results we can determine the tradeoff between the closeness of the approximation and the amount of resources needed to compute the approximation. It also allows us to compare the complexity of NP-approximation problems to the complexity of problems that do not involve optimization or approximation. We start by showing that with enough queries, an NP function can approximate MaxClique.

Lemma 8 Let $k(n)$ be a polynomial-time computable function such that $k(n) < n$. Then, there exists a multivalued function in $\mathrm{NPF}_b^{\mathrm{SAT}[\lceil \log \lceil \log_{k(n)} n \rceil \rceil]}$ that $k(n)$-approximates MaxClique.

Proof: Given a graph G with n vertices, we divide the numbers from 1 to n into intervals that are separated by a factor of $k(n)$:

$$[1, k(n)) \,, \left[k(n), k(n)^2\right) , \left[k(n)^2, k(n)^3\right) , \ldots$$

Since there are $\lceil \log_{k(n)} n \rceil$ intervals, we can use $\lceil \log \lceil \log_{k(n)} n \rceil \rceil$ queries to do binary search and find the interval $[x, k(n)x)$ that contains $\omega(G)$. The endpoint x $k(n)$-approximates $\omega(G)$. So, an NP machine can guess a subgraph X with x vertices, check that X is a clique, and output the vertices of X. □

The procedure described in the proof above obviously works for all NP-optimization problems where the objective function is polynomially bounded and for problems in poly-APX by first applying the polynomial-factor approximation algorithm. One might wonder if the nondeterminism is necessary. In the next lemma, we show that a deterministic machine with many more queries cannot achieve even a coarse approximation of MaxClique.

Lemma 9 Let $q(n) \in O(\log n)$ be a polynomial-time computable function. There exists an $\epsilon > 0$, such that if some function in $\mathrm{PF}^{\mathrm{SAT}[q(n)]}$ can n^ϵ-approximate MaxClique, then P = NP.

Proof Sketch: Use the same ϵ in the theorem by Arora, Lund, Motwani, Sudan and Szegedy which showed that no polynomial-time algorithm can n^ϵ-approximate ω unless P = NP.[3] Given a $\mathrm{PF}^{\mathrm{SAT}[q(n)]}$ machine that n^ϵ-approximates MaxClique, we search the entire oracle computation tree in polynomial time. One of the outputs is a subgraph of G that is a clique with more than n^ϵ vertices. Since we can check each output produced in the oracle computation tree for this property, we can n^ϵ-approximate $\omega(G)$ in polynomial time. Thus, P = NP. □

It appears that we need to have nondeterminism to $k(n)$-approximate MaxClique. The next question is whether we need to have $\log\log_{k(n)} n$ queries to SAT. To do this, we must first show that every $\mathrm{NPF}_b^{\mathrm{SAT}[q(n)]}$ computation can be simulated by one which asks the queries deterministically. We use the notation introduced by Köbler and Thierauf $\mathrm{NPF}//\mathrm{PF}^{\mathrm{SAT}[q(n)]}$ to denote these functions.[28]

Lemma 10 Let $q(n) \in O(\log n)$ be a polynomial-time computable function. Then,

$$\mathrm{NPF}_b^{\mathrm{SAT}[q(n)]} = \mathrm{NPF}//\mathrm{PF}^{\mathrm{SAT}[q(n)]}.$$

Proof Sketch: This is a mind change proof. Mind changes were used to show that $\mathrm{P}^{\mathrm{SAT}[k]} = \mathrm{P}^{\mathrm{SAT}\|[2^k-1]}$ which improves upon Hemachandra's result mentioned in the previous section. Here we use $q(n)$ queries to determine the maximum number of mind changes made by the $\mathrm{NPF}_b^{\mathrm{SAT}[q(n)]}$ machine. Given this additional information, an NPF machine can guess a set of computation paths and verify that they represent the maximum number of mind changes. This will produce the set of all queries to SAT that are answered "yes" in the entire computation tree. Then, the NPF machine can simulate the $\mathrm{NPF}_b^{\mathrm{SAT}[q(n)]}$ machine directly by oracle replacement. □

Corollary 11 MaxClique is $\leq_{\mathrm{mv}}^{\mathrm{P}}$-complete for $\mathrm{NPF}_b^{\mathrm{SAT}[O(\log n)]}$.

Proof Sketch: MaxClique(G) can be computed by an $\mathrm{NPF}_b^{\mathrm{SAT}[O(\log n)]}$ machine that uses the $\log n$ queries to find $\omega(G)$ then guesses a clique of that size. Now, let f be a multivalued function computed by an $\mathrm{NPF}_b^{\mathrm{SAT}[c \log n]}$ machine. We use Lemma 10 to decompose the computation into a deterministic query phase and a nondeterministic phase. The oracle computation tree of the deterministic phase has at most polynomially many query nodes, each node labelled by a query to SAT. We reduce each of these formulas to CLIQUE and obtain the graphs $G_1, \ldots, G_i$. Each leaf of the oracle computation tree is followed by a nondeterministic computation which we turn into a formula using

Karp's reduction (making the machine accept if the computation produces an output). These formulas are again reduced to CLIQUE, producing the graphs $H_1, \ldots, H_j$. Finally, we combine all the graphs $G_1, \ldots, G_i$ and $H_1, \ldots, H_j$ to form one big graph $\mathcal{G}$ so that the largest clique in $\mathcal{G}$ has components that are maximum cliques of the subgraphs. Now, the sizes of the maximum cliques in $G_1, \ldots, G_i$ will tell us how the oracle queries were answered. So, we can find the leaf k, where the actual NP computation takes place. Then, the vertices of the largest clique in H_k will give us a particular accepting path of the computation, which also represents a path where the original machine produced an output. By simulating the NP computation along this computation path, we can recover this output.

The procedure described above is an $\leq^{\mathrm{P}}_{\mathrm{mv}}$-reduction to MaxClique because the construction of $\mathcal{G}$ is the T_1 function and recovering the output is the T_2 function. □

We can prove a similar theorem for $k(n)$-approximating MaxClique.[7] In the proof sketch below, we omit most of arithmetic.

Theorem 12 There exists ϵ, with $0 < \epsilon < 1$, such that for any non-decreasing polynomial-time computable function $q(n) \in O(\log n)$, if h is a function which $k(n)$-approximates MaxClique, then every $f \in \mathrm{NPF}_b^{\mathrm{SAT}[q(n)]}$ strongly $\leq^{\mathrm{P}}_{\mathrm{mv}}$-reduces to h, where $q(n) = \log\log_{k(n)} n - c$ and $c = 1 + \log(1 + 1/\epsilon)$.

Proof Sketch: Here ϵ is the same ϵ described in Lemma 9. First we need to show that the following promise problem is complete for $\mathrm{NPF}_b^{\mathrm{SAT}[q(n)]}$: given $F_1, \ldots, F_r$ with $r = 2^{q(n)} - 1$, find a satisfying assignment of F_z where $z = \max\{i \mid F_i \in \mathrm{SAT}\}$ under the promise that $F_{i+1} \in \mathrm{SAT}$ implies $F_i \in \mathrm{SAT}$. Then, we use the gap-creating reduction from SAT to CLIQUE (see Section 3.2) and create the graphs $G_1, \ldots, G_r$ from $F_1, \ldots, F_r$. We follow the construction of Chang, Gasarch and Lund to build a big graph H with n vertices and numbers $y_1, \ldots, y_r$ such that $k(n)y_i < y_{i+1}$ and

$$y_z \leq \omega(G) < y_{z+1} \iff z = \max\{i \mid F_i \in \mathrm{SAT}\}.$$

Now, by the construction of H, we can guarantee that for any subgraph X of H which $k(n)$-approximates MaxClique(H), X must include enough vertices from G_z so that $X \cap G_z$ approximates MaxClique(G_z) sufficiently closely so that the vertices of a maximum clique of G_z can be recovered in polynomial time (using some PCP magic). The maximum clique of G_z can be used to find a satisfying assignment of F_z which can then be used to compute a value output by the original $\mathrm{NPF}_b^{\mathrm{SAT}[q(n)]}$ function. □

Combining Lemma 8 and Theorem 12, we can show that 2-approximating MaxClique is $\leq^{\mathrm{P}}_{\mathrm{mv}}$-complete for the appropriate bounded query class.

Corollary 13 2-approximating MaxClique is $\leq^{\mathrm{P}}_{\mathrm{mv}}$-complete for the class $\mathrm{NPF}_b^{\mathrm{SAT}[\log\log n+O(1)]}$.

Using Lemma 8, we can construct a function in $\mathrm{NPF}_b^{\mathrm{SAT}[\log\log n]}$ which 2-approximates Color. Thus, we have the following corollary.

Corollary 14 There is a $\leq^{\mathrm{P}}_{\mathrm{mv}}$-reduction from 2-approximating Color to 2-approximating MaxClique.

In fact, Corollary 14 applies to every NP-optimization problem in poly-APX, because Lemma 8 holds for these problems. We are tempted to claim that every function that strongly $\leq^{\mathrm{P}}_{\mathrm{mv}}$-reduces to 2-approximating MaxClique is in $\mathrm{NPF}_b^{\mathrm{SAT}[\log\log n+O(1)]}$. This is not the case for the following technicality. Lemma 8 only shows the existence of a function f in $\mathrm{NPF}_b^{\mathrm{SAT}[\log\log n]}$ that 2-approximates MaxClique. It does not show that "2-approximating MaxClique" itself is a function in $\mathrm{NPF}_b^{\mathrm{SAT}[\log\log n]}$. The distinction is that the function f on input G will output *some* of the 2-approximate cliques in G not necessarily all of them. For most applications, this is a distinction that makes no difference. Mathematically, however, we cannot claim that $g \in \mathrm{NPF}_b^{\mathrm{SAT}[\log\log n+O(1)]}$ even when g strongly $\leq^{\mathrm{P}}_{\mathrm{mv}}$-reduces to 2-approximating MaxClique.

Now, we would like to show that we cannot approximate MaxClique with fewer queries. As in the proof by Crescenzi *et al.*, we rely on the Boolean Hierarchy to help us. It turns out that the *languages* accepted by $\mathrm{NP}_b^{\mathrm{SAT}[q(n)]}$ machines fit precisely in the Boolean Hierarchy. For constant k, the k-th level of the Boolean Hierarchy, BH_k, is the class of all languages that can be expressed as nested differences of k NP languages.[29,30] That is, $L \in \mathrm{BH}_k$ if there exist $L_1, \ldots, L_k \in \mathrm{NP}$ such that $L = L_1 - (L_2 - (\cdots - L_k))$. To define the levels of the Boolean Hierarchy beyond constants, note that in the preceding case, $x \in L$ if and only if $\max\{i \mid x \in L_i\}$ is odd. Also, we can restrict ourselves to NP languages such that $L_{i+1} \subseteq L_i$. Thus, to generalize the Boolean Hierarchy, we say that $L \in \mathrm{BH}_{q(n)}$ if there exists an NP machine N such that on input x of length n

- for $i \geq 1$, $N(x, i+1)$ accepts $\Longrightarrow N(x,i)$ accepts.
- for $i > q(n)$, $N(x,i)$ rejects.
- $x \in L \iff \max\{i \mid N(x,i)$ accepts $\}$ is odd.

The Boolean Hierarchy over $\mathrm{NP}^{\mathrm{NP}}$ can be defined in the same manner by replacing the NP languages with $\mathrm{NP}^{\mathrm{NP}}$ languages. We denote the k-th level of this hierarchy with $\mathrm{BH}_k(\mathrm{NP}^{\mathrm{NP}})$.

Now, we can prove a very tight connection between $\mathrm{NP}_b^{\mathrm{SAT}[q(n)]}$ and the Boolean Hierarchy.

Lemma 15 Let $q(n) \in O(\log n)$ be a polynomial-time computable function. Then,

$$\mathrm{BH}_{2^{q(n)+1}-1} = \mathrm{NP}_b^{\mathrm{SAT}[q(n)]}.$$

Proof: By Lemma 10, the $\mathrm{NP}_b^{\mathrm{SAT}[q(n)]}$ computation can be decomposed into a $\mathrm{NP}//\mathrm{PF}^{\mathrm{SAT}[q(n)]}$ computation. Then, the equality follows from a theorem by Köbler and Thierauf.[28] In their terminology, $\mathrm{BH}_{2^{q(n)+1}-1} = \mathrm{NP}//\mathrm{PF}^{\mathrm{SAT}[q(n)]}$ is written as

$$\mathrm{NP}(2^{q(n)+1} - 1) = \mathrm{NP}//\mathrm{OptP}[q(n)].$$

Their theorem only states the result for constant $q(n)$, but the proof generalizes easily for all $q(n) \in O(\log n)$. $\square$

Theorem 16 Let α be constant such that $0 < \alpha < 1$ and let $q(n) < \alpha \log n$ be a polynomial-time computable function. Then, $\mathrm{NPF}_b^{\mathrm{SAT}[q(n)]} = \mathrm{NPF}_b^{\mathrm{SAT}[q(n)+1]}$ implies that

$$\mathrm{PH} \subseteq \mathrm{BH}_{2^{q(n)+1}-1}(\mathrm{NP}^{\mathrm{NP}}) = \mathrm{NP}_b^{(\mathrm{NP}^{\mathrm{NP}})[q(n)]}.$$

Proof Sketch: First note that $\mathrm{NPF}_b^{\mathrm{SAT}[q(n)]} = \mathrm{NPF}_b^{\mathrm{SAT}[q(n)+1]}$ implies that the language classes $\mathrm{NP}_b^{\mathrm{SAT}[q(n)]} = \mathrm{NP}_b^{\mathrm{SAT}[q(n)+1]}$. This step is not entirely obvious since the characteristic function of a language in $\mathrm{NP}_b^{\mathrm{SAT}[q(n)]}$ is not immediately a function in $\mathrm{NPF}_b^{\mathrm{SAT}[q(n)]}$ (because $\mathrm{NPF}_b^{\mathrm{SAT}[q(n)]}$ functions must be total). However, we can compute the characteristic function of an $\mathrm{NP}_b^{\mathrm{SAT}[q(n)]}$ language using $q(n) + 1$ queries to SAT, because

$$\mathrm{NP}_b^{\mathrm{SAT}[q(n)]} = \mathrm{BH}_{2^{q(n)+1}-1} \subseteq \mathrm{P}^{\mathrm{SAT}[q(n)+1]}.$$

Then, by our assumption the characteristic function of every $\mathrm{NP}_b^{\mathrm{SAT}[q(n)]}$ language and its complement are computable in $\mathrm{NPF}_b^{\mathrm{SAT}[q(n)]}$. Therefore, $\mathrm{NP}_b^{\mathrm{SAT}[q(n)]}$ is closed under complementation and the Boolean Hierarchy collapses at level $2^{q(n)+1} - 1$. The deepest collapse of PH due to this collapse of the Boolean Hierarchy can be found in papers by Chang and Kadin and by Beigel, Chang and Ogiwara.[11,12] The proofs in these papers only mention the constant levels of the Boolean Hierarchy, but they can be extended to the more general case using the observations of Wagner.[18] $\square$

Corollary 17 Suppose that MaxClique $\leq_{\rm mv}^{\rm P}$-reduces to 2-approximating MaxClique. Then $\mathrm{PH} \subseteq \mathrm{NP}_b^{(\mathrm{NP}^{\mathrm{NP}})[\log\log n+O(1)]}$.

One advantage of having this machine model is that we can consider situations where MaxClique could reduce to 2-approximating MaxClique. The following corollary also suggests that we will not be able to prove that MaxClique $\leq_{\rm mv}^{\rm P}$-reduces to 2-approximating MaxClique implies $\mathrm{P} = \mathrm{NP}$.

Corollary 18 If NP = co-NP, then there exists a strong $\leq_{\rm mv}^{\rm P}$-reduction from MaxClique to 2-approximating MaxClique.

Proof: If NP = co-NP, then an NP machine does not need to use a SAT oracle. Thus, $\mathrm{NPF}_b^{\mathrm{SAT}[poly]} = \mathrm{NPF}$. In particular, $\mathrm{NPF}_b^{\mathrm{SAT}[\log n]} = \mathrm{NPF}_b^{\mathrm{SAT}[\log\log n]}$, so we can derive the reduction we need from Corollary 11 and Theorem 12. $\square$

In summary, we have combined the proof techniques of Chang-Gasarch-Lund, Khanna-Motwani-Sudan-Vazirani and Crescenzi-Kann-Silvestri-Trevisan and made heavy use of structural complexity theory to obtain the following equivalences:

- $\mathrm{PF}^{\mathrm{SAT}[\log\log n+O(1)]}$ captures the complexity of finding the *size* of a 2-approximate clique.
- $\mathrm{NPF}_b^{\mathrm{SAT}[\log\log n+O(1)]}$ captures the complexity of finding the *vertices* of a 2-approximate clique.

Thus, the difference between finding the size and finding the vertices is exactly nondeterminism. In retrospect, this makes perfect sense since nondeterminism allows you to guess certificates. The proof, however, is much less obvious.

5 Discussion

There are several observations we can make at this point. The first is that it is possible to have a machine-based resource-bounded complexity measure for NP-approximation problems, remarks by Papadimitriou and Yannakakis not withstanding. In fact, this should hardly be surprising, since much of the study of NP-completeness and computational complexity involves embedding computations into natural objects. What we've had to do here is embed many NP-computations into one approximation problem. This has not been possible for problems like CLIQUE until the non-approximability results have been proven using probabilistically checkable proofs. Indeed, it would be more surprising if NP-approximation problems could only be modelled syntactically and not computationally, since that would run counter to most of complexity theory where the typical complexity classes have both computational and syntactic models.

A second observation is that we obtained a cleaner model by embracing rather than rejecting the "multivaluedness" of NP-approximation problems. For example, Chen and Toda used the isolation lemma to find the vertices of a single maximum clique in randomized polynomial time with polynomially many parallel queries to SAT.[31] However, the isolation lemma produces a unique solution with only low probability, so we have to repeat the procedure many times to find the maximum clique with good enough probability.[32,33,34] While it is possible to determine the tradeoff between the probability of success and the number of queries, combining this analysis with the isolation lemma seems quite painful.[13,14] As a result, the repeated use of the isolation lemma makes it difficult to count the number of queries needed to approximate MaxClique. Thus, insisting on a single output makes it difficult to distinguish the complexities of computing MaxClique exactly and that of approximating MaxClique.

Our final observation is that the complexity of NP-approximation problems depend not just on the P versus NP question, but also on the NP versus co-NP question. While it is intuitively obvious that finding the largest clique should be harder than finding an approximate clique, this is only the case if NP is not closed under complementation (q.v. Corollary 18). The connection between NP-approximation problems and the Boolean Hierarchy becomes clearer when one thinks of the Boolean Hierarchy as a generalization of the complementation operation (sometimes called "hardware over NP"). It is nice to see that theorems about the Boolean Hierarchy have applications in other areas — it has been many years since the Boolean Hierarchy has been defined and many more since people have used the number of queries as a complexity measure.[35]

References

1. U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy. Interactive proofs and the hardness of approximating clique. *Journal of the ACM*, 43(2):268–292, March 1996.
2. S. Arora and S. Safra. Probabilistic checking of proofs: A new charac-

terization of NP. *Journal of the ACM*, 45(1):70–122, January 1998.
3. S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and hardness of approximation problems. *Journal of the ACM*, 45(3):501–555, May 1998.
4. C. Lund and M. Yannakakis. On the hardness of approximating minimization problems. *Journal of the ACM*, 41(5):960–981, September 1994.
5. R. Chang and W. I. Gasarch. On bounded queries and approximation. In *Proceedings of the IEEE Symposium on Foundations of Computer Science*, pages 547–556, November 1993.
6. R. Chang, W. I. Gasarch, and C. Lund. On bounded queries and approximation. *SIAM Journal on Computing*, 26(1):188–209, February 1997.
7. R. Chang. On the query complexity of clique size and maximum satisfiability. *Journal of Computer and System Sciences*, 53(2):298–313, October 1996.
8. S. Khanna, R. Motwani, M. Sudan, and U. Vazirani. On syntactic versus computational views of approximability. *SIAM Journal on Computing*, 28(1):164–191, 1998.
9. P. Crescenzi and L. Trevisan. On approximation scheme preserving reducibility and its applications. *Theory of Computing Systems*, 33(1):1–16, 2000.
10. P. Crescenzi, V. Kann, R. Silvestri, and L. Trevisan. Structure in approximation classes. *SIAM Journal on Computing*, 28(5):1759–1782, 1999.
11. R. Chang and J. Kadin. The Boolean hierarchy and the polynomial hierarchy: A closer connection. *SIAM Journal on Computing*, 25(2):340–354, April 1996.
12. R. Beigel, R. Chang, and M. Ogiwara. A relationship between difference hierarchies and relativized polynomial hierarchies. *Mathematical Systems Theory*, 26(3):293–310, July 1993.
13. R. Chang, J. Kadin, and P. Rohatgi. On unique satisfiability and the threshold behavior of randomized reductions. *Journal of Computer and System Sciences*, 50(3):359–373, June 1995.
14. P. Rohatgi. Saving queries with randomness. *Journal of Computer and System Sciences*, 50(3):476–492, June 1995.
15. J. Hartmanis and R. E. Stearns. On the computational complexity of algorithms. *Transactions of the AMS*, 117:285–306, 1965.
16. R. E. Stearns, J. Hartmanis, and P. M. Lewis II. Hierarchies of memory limited computations. In *Proceedings of the Sixth Annual Symposium on Switching Circuit Theory and Logical Design*, pages 179–190, 1965.
17. C. H. Papadimitriou and M. Yannakakis. Optimization, approximation,

and complexity classes. In *ACM Symposium on Theory of Computing*, pages 229–234, 1988.

18. K. Wagner. Bounded query computations. In *Proceedings of the 3rd Structure in Complexity Theory Conference*, pages 260–277, June 1988.
19. R. V. Book, T. J. Long, and A. L. Selman. Quantitative relativizations of complexity classes. *SIAM Journal on Computing*, 13:461–487, 1984.
20. M. W. Krentel. The complexity of optimization problems. *Journal of Computer and System Sciences*, 36(3):490–509, 1988.
21. A. Amir, R. Beigel, and W. I. Gasarch. Some connections between bounded query classes and non-uniform complexity. In *Proceedings of the 5th Structure in Complexity Theory Conference*, pages 232–243, 1990.
22. A. Panconesi and D. Ranjan. Quantifiers and approximation. *Theoretical Computer Science*, 107(1):145–163, January 1993.
23. J. Kadin. The polynomial time hierarchy collapses if the Boolean hierarchy collapses. *SIAM Journal on Computing*, 17(6):1263–1282, December 1988.
24. L. Hemachandra. The strong exponential hierarchy collapses. *Journal of Computer and System Sciences*, 39(3):299–322, 1989.
25. S. Mahaney. Sparse complete sets for NP: Solution of a conjecture of Berman and Hartmanis. *Journal of Computer and System Sciences*, 25(2):130–143, 1982.
26. R. Beigel. Bounded queries to SAT and the Boolean hierarchy. *Theoretical Computer Science*, 84(2):199–223, July 1991.
27. M. Ogiwara and O. Watanabe. On polynomial time truth-table reducibility of NP sets to sparse sets. *SIAM Journal on Computing*, 20(3):471–483, 1991.
28. J. Köbler and T. Thierauf. Complexity-restricted advice functions. *SIAM Journal on Computing*, 23(2):261–275, April 1994.
29. J. Cai, T. Gundermann, J. Hartmanis, L. Hemachandra, V. Sewelson, K. Wagner, and G. Wechsung. The Boolean hierarchy I: Structural properties. *SIAM Journal on Computing*, 17(6):1232–1252, December 1988.
30. J. Cai, T. Gundermann, J. Hartmanis, L. Hemachandra, V. Sewelson, K. Wagner, and G. Wechsung. The Boolean hierarchy II: Applications. *SIAM Journal on Computing*, 18(1):95–111, February 1989.
31. Z. Z. Chen and S. Toda. On the complexity of computing optimal solutions. Unpublished manuscript.
32. L. G. Valiant and V. V. Vazirani. NP is as easy as detecting unique solutions. *Theoretical Computer Science*, 47(1):85–93, 1986.
33. K. Mulmuley, U. V. Vazirani, and V. V. Vazirani. Matching is as easy

as matrix inversion. *Combinatorica*, 7(1):105–113, 1987.

34. S. Chari, P. Rohatgi, and A. Srinivasan. Randomness-optimal element isolation, with applications to perfect matching and related problems. *SIAM Journal on Computing*, 24(5):1036–1050, October 1995.
35. C.H. Papadimitriou and S.K. Zachos. Two remarks on the power of counting. Technical Report MIT/LCS/TM-228, Massachusetts Institute of Technology, Laboratory for Computer Science, August 1982.