ECCC
TR97-035
FTP: ftp.eccc.uni-trier.de:/pub/eccc/
WWW: http://www.eccc.uni-trier.de/eccc/
Email: ftpmail@ftp.eccc.uni-trier.de with subject 'help eccc'

Bounded Queries, Approximations and the Boolean Hierarchy

Richard Chang[†]

Department of Computer Science and Electrical Engineering
University of Maryland Baltimore County

July 27, 1997

Abstract

This paper introduces a new model of computation for describing the complexity of NP-approximation problems. The results show that the complexity of NP-approximation problems can be characterized by classes of multi-valued functions computed by nondeterministic polynomial time Turing machines with a bounded number of oracle queries to an NP-complete language. In contrast to the bounded query classes used in previous studies, the machines defined here solve NP-approximation problems by providing the witness to an approximate solution — not just by estimating the size of the objective function. Furthermore, the introduction of this new model of computation is justified in three ways. First, the definition of these complexity classes is robust. Second, NP-approximation problems are complete problems for these complexity classes. This paper concentrates on the Traveling Salesman Problem and the MaxClique Problem. However, these results are general enough to extend to problems such as Graph Coloring and Maximum Satisfiability using existing techniques in the literature. Finally, the utility of this model of computation is demonstrated by proving some new upward collapse results for NP-approximation problems that would be difficult to achieve without the machinery provided by the model.

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

1 Introduction

In this paper, we introduce a new model of computation for describing the complexity of NP-approximation problems. The model used here is a variation of the well-studied bounded query classes which was first used by Krentel [Kre88] to distinguish the complexity of different NP-optimization problems. For example, Krentel showed that a deterministic polynomial time Turing machine using polynomially many oracle queries to an NP-complete language — a $\mathrm{PF}^{\mathrm{SAT}[n^{O(1)}]}$ machine in our terminology — can compute the length of the shortest traveling salesman tour in an undirected graph. In fact, Krentel showed that this problem is complete for $\mathrm{PF}^{\mathrm{SAT}[n^{O(1)}]}$. On the other hand, the size of the largest clique in a graph can be computed with only $O(\log n)$ queries to SAT.[1] Thus, the number of queries to the SAT oracle is a complexity measure which distinguishes the difference between the complexity of the Traveling Salesman Problem (TSP) and that of the MaxClique problem. The connection between bounded query classes and the complexity of NP-approximation problems was established by Chang, Gasarch and Lund [CGL97, Cha96]. Their results show a tight trade-off between the number of queries to SAT needed to approximate the size of the largest clique and the closeness of the approximation — in general, closer approximations require more queries.

In contrast to previous studies, in this paper we require that the machines solve the NP-approximation problems by providing a witness to an approximate solution — not just by estimating the size of the objective function. For example, in the case of TSP we require that the machines report a tour of the vertices of the graph that has approximately good length, instead of simply reporting the approximate length. Similarly, an approximate solution to MaxClique consists of the vertices of a clique that is guaranteed to be close in size to the largest clique. This is arguably a more natural formulation of NP-approximation problems since algorithms for NP-approximation problems are generally required to find such witnesses. The results in this paper show that the complexity classes defined by NP machines with a bounded number of queries to the SAT oracle provide a good complexity measure for such NP-approximation problems. Note that a $\mathrm{PF}^{\mathrm{SAT}[n^{O(1)}]}$ machine can also find the optimum TSP tour by expending another n queries to find the ordering of the vertices of the optimum tour after it has determined the length of the optimum tour. However, using such a strategy one would also need to use $n^{O(1)}$ queries to SAT to find the largest clique in a graph or even just a 2-approximate clique. Thus, the complexity classes defined by the $\mathrm{PF}^{\mathrm{SAT}[q(n)]}$ machines fail to distinguish the complexity of these problems. Furthermore, our results show that finding a 2-approximate clique in a graph cannot be complete for $\mathrm{PF}^{\mathrm{SAT}[n^{O(1)}]}$ unless the Polynomial Hierarchy (PH) collapses.

We justify the introduction of our new model of computation in three ways. First, we show in Section 3 that the definition of these complexity classes is fairly robust in that there are several equivalent definitions. Second, we show in Section 4 that several NP-approximation problems are actually complete problems for these complexity classes. In this paper, we concentrate on TSP and MaxClique. However, these results are general enough to extend to problems such as Graph Coloring and Maximum Satisfiability using existing techniques in the literature [CGL97, Cha96, KMSV94]. Finally, in Section 6 we demonstrate the utility of this model of computation by proving some new upward collapse results for NP-approximation problems that would be difficult to achieve without the machinery provided by the model.

These upward collapse results show that if two NP-approximation problems of lower, but seemingly different, complexity are inter-reducible, then problems at a higher level are also inter-

[1] Without loss of generality, any NP-complete language can be used as the oracle here. We choose SAT, the set of satisfiable Boolean formulas.

reducible. For example, we show that

$$\begin{aligned}&\textsc{MaxClique} \leq^{\mathrm{P}}_{\mathrm{m}} (\log n)\text{-approximating } \textsc{MaxClique}\\ &\quad\Longrightarrow \text{TSP} \leq^{\mathrm{P}}_{\mathrm{m}} \text{2-approx. TSP}\\ &\quad\Longrightarrow \text{TSP} \equiv^{\mathrm{P}}_{\mathrm{m}} \text{2-approx. TSP} \equiv^{\mathrm{P}}_{\mathrm{m}} \textsc{MaxClique} \equiv^{\mathrm{P}}_{\mathrm{m}} (\log n)\text{-approx. } \textsc{MaxClique}.\end{aligned}$$

This theorem shows that if MaxClique is easier than we might reasonably suppose (i.e., solving MaxClique is only as hard as approximating MaxClique within a factor of $\log n$), then the Traveling Salesman Problem (TSP) is also easier than we might suppose. It has been previously shown that if PH does not collapse, then MaxClique does not reduce to $\log n$-approximating MaxClique [CKST95]. However, this result does not have any direct implication regarding complexity of TSP. Other upward collapse results we prove include

$$\begin{aligned}&\text{2-approximating } \textsc{MaxClique} \leq^{\mathrm{P}}_{\mathrm{m}} \text{2-approximating } \textsc{Set Cover}\\ &\quad\Longrightarrow \text{TSP} \equiv^{\mathrm{P}}_{\mathrm{m}} \text{2-approx. TSP} \equiv^{\mathrm{P}}_{\mathrm{m}} \textsc{MaxClique} \equiv^{\mathrm{P}}_{\mathrm{m}} \text{2-approx. } \textsc{MaxClique},\end{aligned}$$

$$\begin{aligned}&\textsc{MaxClique} \leq^{\mathrm{P}}_{\mathrm{m}} \text{2-approximating } \textsc{MaxClique}\\ &\quad\Longrightarrow \text{TSP} \equiv^{\mathrm{P}}_{\mathrm{m}} (1+n^{-\log n})\text{-approximating TSP}.\end{aligned}$$

The proofs of these upward collapses reveal some intricate connections among NP-approximation problems, nondeterministic bounded query classes and the Boolean Hierarchy. Even though the statement of the upward collapse theorems do not mention the Boolean Hierarchy or even Turing machines, the only way that we can prove these upward collapses is to examine the consequences of collapsing the Boolean Hierarchy and the nondeterministic bounded query hierarchy. We use $\mathrm{NPF}^{\mathrm{SAT}[q(n)]}$ to denote the class of total multi-valued functions computed by nondeterministic Turing machines which ask at most $q(n)$ queries to the SAT oracle in the entire nondeterministic computation tree. For example, in Section 4 we prove that 2-approximating MaxClique is complete for the class $\mathrm{NPF}^{\mathrm{SAT}[O(\log\log n)]}$, meaning (quite naturally) that there exists a multi-valued function f in $\mathrm{NPF}^{\mathrm{SAT}[O(\log\log n)]}$ which 2-approximates MaxClique and every function in the class reduces to any function that 2-approximates MaxClique. Since there is a correspondence between the nondeterministic bounded query hierarchy and NP-approximation problems, an upward collapse theorem for the nondeterministic bounded query hierarchy leads to an upward collapse of NP-approximation problems. However, it is difficult to obtain upward collapses beyond a constant number of levels. For example, one can easily show that

$$\mathrm{NPF}^{\mathrm{SAT}[\log\log n]} = \mathrm{NPF}^{\mathrm{SAT}[\log\log n+1]} \Longrightarrow \forall k,\ \mathrm{NPF}^{\mathrm{SAT}[\log\log n+k]} \subseteq \mathrm{NPF}^{\mathrm{SAT}[\log\log n]},$$

but proving that $\mathrm{NPF}^{\mathrm{SAT}[2\log\log n]} \subseteq \mathrm{NPF}^{\mathrm{SAT}[\log\log n]}$ under this assumption is very difficult. In fact, it remains an open question. Instead, we are able to show that

$$\mathrm{NPF}^{\mathrm{SAT}[\log\log n]} = \mathrm{NPF}^{\mathrm{SAT}[\log\log n+1]} \Longrightarrow \mathrm{NPF}^{\mathrm{SAT}[n^{O(1)}]} \subseteq \mathrm{NPF}^{\mathrm{SAT}[\log^2 n]}, \tag{1}$$

which is sufficient to prove that TSP $\equiv^{\mathrm{P}}_{\mathrm{m}} (1+n^{-\log n})$-approximating TSP.

The upward collapse in (1) depends on a surprising connection between the Boolean Hierarchy and nondeterministic bounded query classes. It has been shown previously [Cha94] that a collapse of the $\mathrm{NPF}^{\mathrm{SAT}[q(n)]}$ classes implies a collapse of the Boolean Hierarchy. In this paper, we show that the converse is also true. For example, for all constants k

$$\mathrm{BH}_{2^k} = \mathrm{coBH}_{2^k} \Longrightarrow \mathrm{NPF}^{\mathrm{SAT}[n^{O(1)}]} \subseteq \mathrm{NPF}^{\mathrm{SAT}[k]}.$$

This result is surprising because it shows that a collapse of a language hierarchy causes a collapse of a function hierarchy. In contrast, it possible that the Boolean Hierarchy collapses, but the

deterministic bounded query function hierarchy $\mathrm{PF}^{\mathrm{SAT}[k]}$ does not (for example, if NP = coNP and P $\neq$ NP). The connection between the Boolean Hierarchy and the $\mathrm{NPF}^{\mathrm{SAT}[q(n)]}$ classes is based upon the very recent work of Buhrman and Fortnow [BF96] who showed that

$$\mathrm{BH}_2 = \mathrm{coBH}_2 \Longrightarrow \mathrm{P}^{\mathrm{SAT}[n^{O(1)}]} \subseteq \mathrm{BH}_2.$$

Our theorems in Section 5 are generalizations of their work.

Finally, the results in this paper also provide additional evidence for an infinite Boolean Hierarchy. The usual argument that the Boolean Hierarchy has infinitely many distinct levels assumes that PH does not collapse [Kad88, BCO93, CK96, Wag88, Wag90]. Using the results in this paper, we show that if MaxClique does not reduce to 2-approximating MaxClique, then the Boolean Hierarchy has infinitely many levels. Furthermore, the assumption that MaxClique does not reduce to 2-approximating MaxClique is weaker and arguably more natural than the assumption that PH does not collapse.

2 Preliminaries

The Boolean Hierarchy is a generalization of the class D^{P} defined by Papadimitriou and Yannakakis [PY82]. For constant k, the kth level of the Boolean Hierarchy can be defined simply as nested differences of NP languages [CGH+88, CGH+89].

Definition 1 For constant k, we use BH_k and coBH_k to denote the kth levels of the Boolean Hierarchy, defined as follows:

$$\begin{aligned}
\mathrm{BH}_1 &= \mathrm{NP}, \\
\mathrm{BH}_2 &= \mathrm{D}^{\mathrm{P}} = \{L_1 - L_2 \mid L_1, L_2 \in \mathrm{NP}\} \\
\mathrm{BH}_{k+1} &= \{L_1 - L_2 \mid L_1 \in \mathrm{NP} \text{ and } L_2 \in \mathrm{BH}_k\}, \\
\mathrm{coBH}_k &= \{L \mid \overline{L} \in \mathrm{BH}_k\}.
\end{aligned}$$

For non-constant levels of the Boolean Hierarchy, we should not simply define a language in $\mathrm{BH}_{r(n)}$ to be the nested differences of $r(n)$ NP languages. This is because the running times of the machines recognizing the $r(n)$ languages might not be bounded by a single polynomial. Thus, we use a single NP machine to define each language in $\mathrm{BH}_{r(n)}$.[2]

Definition 2 Let $r(n)$ be a monotonically increasing polynomial-time computable function such that $r(n) \leq n^{\epsilon}$ for some constant $\epsilon < 1$. A language $L \in \mathrm{BH}_{r(n)}$ if there exists an NP machine N such that

$$x \in L \iff \max(\{i \mid 1 \leq i \leq r(n) \text{ and } N(x,i) \text{ accepts}\} \cup \{0\}) \text{ is odd.}$$

Also, $\overline{L} \in \mathrm{BH}_{r(n)}$ implies that $L \in \mathrm{coBH}_{r(n)}$.

Elementary results on the Boolean Hierarchy show that $\mathrm{BH}_k \cup \mathrm{coBH}_k \subseteq \mathrm{BH}_{k+1} \cap \mathrm{coBH}_{k+1}$ and that a constant upward collapse theorem holds [CGH+88, CGH+89]:

$$\mathrm{BH}_k = \mathrm{coBH}_k \Longrightarrow \mathrm{BH}_k = \mathrm{BH}_{k+1} \Longrightarrow \forall k' > k,\ \mathrm{BH}_k = \mathrm{BH}_{k'}.$$

There are several ways to define complete languages for the Boolean Hierarchy. Depending upon the application, we will use $\mathrm{BL}_{r(n)}$ or $\mathrm{ODD}^{\mathrm{SAT}}_{r(n)}$ defined below to be the canonical $\leq^{\mathrm{P}}_{\mathrm{m}}$-complete language for $\mathrm{BH}_{r(n)}$.

[2] See [Wag88, Wag90] for a thorough discussion on this matter.

Definition 3 Let $r(n)$ be a monotonically increasing polynomial-time computable function such that $r(n) \leq n^\epsilon$ for some constant $\epsilon < 1$. We define the languages $\mathrm{BL}_{r(n)}$ as follows:

$$\begin{aligned}
\mathrm{BL}_1 &= \mathrm{SAT} \\
\mathrm{BL}_{2r(n)} &= \{\langle x_1, \ldots, x_{2r(n)}\rangle \mid \langle x_1, \ldots, x_{2r(n)-1}\rangle \in \mathrm{BL}_{2r(n)-1} \text{ and } x_{2r(n)} \in \overline{\mathrm{SAT}}\} \\
\mathrm{BL}_{2r(n)+1} &= \{\langle x_1, \ldots, x_{2r(n)+1}\rangle \mid \langle x_1, \ldots, x_{2r(n)}\rangle \in \mathrm{BL}_{2r(n)} \text{ or } x_{2r(n)+1} \in \mathrm{SAT}\} \\
\mathrm{coBL}_{r(n)} &= \overline{\mathrm{BL}_{r(n)}} \\
\mathrm{ODD}^{\mathrm{SAT}}_{r(n)} &= \{\langle x_1, \ldots, x_{r(n)}\rangle \mid \|\{i \mid (1 \leq i \leq m) \wedge (x_i \in \mathrm{SAT})\}\| \text{ is odd.}\}
\end{aligned}$$

The notation $\langle x_1, \ldots, x_{r(n)}\rangle$ is shorthand for a tuple $\langle x_1, \ldots, x_m\rangle$ where n is the length of the tuple $|\langle x_1, \ldots, x_m\rangle|$ and $m = r(n)$.

Definition 4 Let $q(n)$ be a polynomial-time computable function. We use $\mathrm{PF}^{\mathrm{SAT}[q(n)]}$ to denote the set of functions computed by deterministic polynomial-time Turing machines which ask at most $q(n)$ queries to the SAT oracle on inputs of length n. When the queries are made in parallel, we use the notation $\mathrm{PF}^{\mathrm{SAT}[q(n)]}_{\mathrm{tt}}$. The classes $\mathrm{P}^{\mathrm{SAT}[q(n)]}$ and $\mathrm{P}^{\mathrm{SAT}[q(n)]}_{\mathrm{tt}}$ are the analogous language classes.

When we work with a $\mathrm{P}^{\mathrm{SAT}[q(n)]}$ or a $\mathrm{PF}^{\mathrm{SAT}[q(n)]}$ computation, it is often useful to consider the computation as an *oracle query tree*. Given a polynomial time machine M which asks at most $q(n)$ queries to an oracle and an input string x of length n, the oracle query tree for $M(x)$ is a full binary tree with depth $q(n)$ and $2^{q(n)} - 1$ internal nodes. Each internal node represents a query made by $M(x)$. The left (right) subtree of the query node represents the computation of $M(x)$ after the query is answered assuming that the answer to the query is NO (YES). Each path from the root to a leaf corresponds to the computation of $M(x)$ using some oracle. We index the $2^{q(n)}$ paths from left to right starting from 0 to $2^{q(n)} - 1$. In this scheme, for a path with index i, the jth bit of the $q(n)$-bit binary representation of i is 1 if and only if the path assumes that the jth query is answered YES. For a given path, we say that a query is *positive* if the path assumes that the query is answered YES; otherwise, we call the query a *negative* query. Whether a query is positive or negative depends on the path under consideration. Every positive query on a path is a negative query on some other path (since we are considering all possible paths).

Let z be the index of the path in the oracle query tree that corresponds to the computation of $M(x)$ using the SAT oracle. We call this path the *correct* path since we will be working almost exclusively with the SAT oracle. For each path with index i in the oracle query tree, we can construct a Boolean formula F_i that is the conjunction of the positive queries on that path. (We define F_0 to be TRUE.) Note that F_z must be satisfiable since z is the index of the correct path and all positive queries on the correct path are indeed satisfiable. Furthermore, note that for all indices $j > z$, $F_j \notin \mathrm{SAT}$ since the assumption that $j > z$ implies that some negative query on path z is a positive query on path j. This query is in fact unsatisfiable, since z is the correct path. Thus, the index of the correct path can also be defined as:

$$z = \max\{\ i \mid 0 \leq i \leq 2^{q(n)} - 1 \text{ and } F_i \in \mathrm{SAT}\ \}.$$

Finding the value of z allows us to recover the entire computation of $M(x)$ without using any oracle queries. This will be a key observation in several proofs.

Definition 5 Let $q(n)$ be a polynomial-time computable function. We use $\mathrm{NPF}^{\mathrm{SAT}[q(n)]}$ to denote the set of *total multi-valued functions* computed by nondeterministic polynomial-time Turing machines which ask at most $q(n)$ queries to the SAT oracle in the entire nondeterministic computation tree on inputs of size n. The class $\mathrm{NP}^{\mathrm{SAT}[q(n)]}$ is the analogous class of languages.

It is not useful to limit the number of oracle queries made by an NP machine on each computation *path* because in that case one query is as powerful as polynomially many queries and $\mathrm{NP}^{\mathrm{SAT}[1]}$ would simply be Σ_2^{P}.[3] Also, a multi-valued function may have several outputs for each input string. It may seem awkward to have to work with multi-valued functions. However, solutions to approximation problems are naturally multi-valued. When we ask for an approximate solution to a problem, we are inherently saying that we do not care which of many possible solutions is produced. In addition, our restriction of the $\mathrm{NPF}^{\mathrm{SAT}[q(n)]}$ classes to total functions is not overly limiting, since every partial function computable by a nondeterministic Turing machine using $q(n)$ queries to SAT has an equivalent total version that is computable using $q(n)+1$ queries to SAT. On inputs where the original function is undefined, the total version outputs a new $\perp$ symbol.

We use the following generalization of many-one reductions for reductions between multi-valued functions.

Definition 6 Let f and g be two multi-valued functions. We say that $f \leq_{\mathrm{m}}^{\mathrm{P}} g$ if there exist two polynomial-time computable functions T_1 and T_2 such that for every input x of f, $T_1(x) = y$ is a string in the domain of g and for every output of z of $g(y)$, $T_2(x,z)$ is an output of $f(x)$.

For example, Graph Coloring $\leq_{\mathrm{m}}^{\mathrm{P}}$ MaxClique. By this we mean that given any graph G, we can produce a graph G' in polynomial-time such that given the vertices of any maximum clique G' (there can be many maximum cliques), we can then produce a coloring of G that uses a minimum number of colors.

Since a $\leq_{\mathrm{m}}^{\mathrm{P}}$-reduction can stretch the length of its output by a polynomial factor, the class $\mathrm{NPF}^{\mathrm{SAT}[O(\log n)]}$ is closed under $\leq_{\mathrm{m}}^{\mathrm{P}}$-reductions whereas the class $\mathrm{NPF}^{\mathrm{SAT}[\log n]}$ is not. This polynomial stretching can also be used to in a padding argument to show that for all $c_1 > c_2 > 0$, every function in $\mathrm{NPF}^{\mathrm{SAT}[c_1 \log n]}$ reduces to some function f' in $\mathrm{NPF}^{\mathrm{SAT}[c_2 \log n]}$. Simply define f' to be the function

$$f'(w) = \begin{cases} f(x) & \text{if } w = x\#0^m, \text{ where } m \geq |x|^{c_1/c_2} \\ 0 & \text{otherwise} \end{cases}$$

Clearly, $f \leq_{\mathrm{m}}^{\mathrm{P}} f'$. Note that the absolute number of queries used to compute f and f' are the same. It is only the number of queries relative to the length of their inputs that differs. Similar closure and padding properties hold for the classes $\mathrm{NPF}^{\mathrm{SAT}[q(n)]}$, where $q(n) = n^{O(1)}$, $q(n) = O(\log^a n)$ and $q(n) = \log\log n + O(1)$.

Definition 7 Let $G = (V, E)$ be an undirected graph with n vertices and let $k(n)$ be an approximation factor such that $\forall n, 1 \leq k(n) \leq n$. We use $\omega(G)$ to denote the size of the largest clique in G. We say that a multi-valued function f $k(\cdot)$-approximates MaxClique if for all graphs G every output of $f(G)$ is a set $X \subseteq V$ such that X is a clique and $|X| \geq \omega(G)/k(n)$.

It is important to note that the approximation factor $k(n)$ is always a function of the number of vertices in G even if G is not the input of the computation under consideration. This is somewhat confusing since we will most often use n to denote the length of the input. Then, if we construct a graph G with m vertices, we need to consider $k(m)$-approximate solutions of G. Thus, we resort to the terminology "$k(\cdot)$-approximation" rather than "$k(n)$-approximation" or k-approximation (which suggests that k is a constant).

[3] Counting queries in the entire nondeterministic computation tree was used by Book, Long and Selman [BLS84, Lon85] in the context of positive relativizations. Wagner [Wag90] has also considered these classes.

Definition 8 Let $G = (V, E)$ be a weighted undirected graph with n vertices and weight function $w : E \to \mathbb{N}$. Without loss of generality, we assume that

$$\sum_{e \in E} w(e) \leq 2^n.$$

A TSP tour in the graph is a cycle that visits each vertex exactly once. The length of a TSP tour is sum of the weights of the edges in the cycle. Let $\textsc{Opt}\text{TSP}(G)$ denote the length of the shortest TSP tour in G and let $k(n)$ be an approximation factor such that $\forall n, 1 \leq k(n) \leq 2^n$. We say that a multi-valued function f $k(\cdot)$-approximates TSP if for all graphs G every output of $f(G)$ is a TSP tour of G with length $\leq k(n)\textsc{Opt}\text{TSP}(G)$.

3 Normal forms for bounded query classes

Although the classes $\text{NPF}^{\text{SAT}[q(n)]}$ are fairly straightforward to define, they are difficult to work with because the queries asked by an $\text{NPF}^{\text{SAT}[q(n)]}$ machine may be the result of nondeterministic computations. In this section, we show that there are equivalent machines which ask all of the queries deterministically. This model of computation is divided into two phases. First, a deterministic phase generates the queries and receives the answers to the queries. This is followed by a second nondeterministic phase. Formally, we define these classes using the // advice operator of Köbler and Thierauf [KT94].

Definition 9 Let $\mathcal{C}$ be a class of functions. Then, $\text{NPF}//\mathcal{C}$ is the set of total multi-valued functions f defined by a function $g \in \mathcal{C}$ and an NP machine N such that the outputs of $f(x)$ are the outputs of $N(x, g(x))$. A language $L \in \text{NP}//\mathcal{C}$ if there exist a $g \in \mathcal{C}$ and an NP machine N such that $x \in L$ if and only if $N(x, g(x))$ accepts.

The following lemmas demonstrate the relationships among several nondeterministic bounded query classes. The techniques used to prove these lemmas include the census trick and the mind change technique. These techniques are fairly common in the literature on bounded query classes [Hem89, Bei91, BCO93, BC97].

Lemma 10 Let $r(n) \in n^{O(1)}$ be a polynomial-time computable function, then

$$\text{NPF}//\text{PF}^{\text{SAT}[r(n)]} \subseteq \text{NPF}^{\text{SAT}[r(n)]} \subseteq \text{NPF}//\text{PF}^{\text{SAT}[2r(n)]}.$$

Proof: The first containment $\text{NPF}//\text{PF}^{\text{SAT}[r(n)]} \subseteq \text{NPF}^{\text{SAT}[r(n)]}$ follows directly from definition. To see that $\text{NPF}^{\text{SAT}[r(n)]} \subseteq \text{NPF}//\text{PF}^{\text{SAT}[2r(n)]}$, consider the queries made by an $\text{NPF}^{\text{SAT}[r(n)]}$ machine N on a particular input x. We classify the queries made by N according to levels. A query q is on level i if q is the ith oracle query on a computation path of N on input x. In this proof we consider only computation paths where the oracle queries are answered correctly. Thus, q must be one of the $r(n)$ queries asked by N on input x. It is not possible for a P machine to enumerate the queries on level 1 (unless P = NP), since N might ask these queries after many nondeterministic moves. On the other hand, an NP machine can simulate N and guess which queries are asked on level 1. However, even an NP machine cannot enumerate the queries that are on level 2 (unless NP = coNP), because doing so requires the answers to the level 1 queries. Nevertheless, if an NP machine is given c_1, the census of the level 1 queries — i.e., the exact number of level 1 queries that are satisfiable — then the NP machine can guess c_1 satisfiable level 1 queries, verify that these strings were indeed queried by N on input x and simulate $N(x)$ to obtain the level 2 queries. The simulation is correct because any level 1 query that is not on the list of c_1 satisfiable level 1 queries

must be unsatisfiable. Similarly, given the census for levels 1 through $i-1$, an NP machine can guess which queries are asked on level i. We use this census strategy and a *linear* search to prove that $\mathrm{NPF}^{\mathrm{SAT}[r(n)]} \subseteq \mathrm{NPF}//\mathrm{PF}^{\mathrm{SAT}[2r(n)]}$.

We construct a $\mathrm{PF}^{\mathrm{SAT}[2r(n)]}$ machine D which will determine for each level i the census c_i — the number of satisfiable level i queries. The machine D simply asks "Are there k satisfiable queries on level i?" for increasing values of k. By the discussion above, this question can be answered by the SAT oracle assuming that the census for levels 1 through $i-1$ are already known — which is true because D determines $c_1, c_2, \ldots$ in order. If D receives YES as an answer, then the original machine N did in fact make k queries on level i. On the other hand, if the answer is NO, then D asked an extra query. However, since we are using a linear search, in this case D knows that c_i is exactly $k-1$. Thus, D asks at most one extra query per level. Observe that the total number of YES answers received by D for all levels must be bounded by $r(n)$, since this number is bounded by the total number of queries that N asked. Furthermore, there are at most $r(n)$ query levels, so the number of NO answers received by D is also bounded by $r(n)$. Therefore, the total number of queries made by D is bounded by $2r(n)$.

Finally, we combine the machine D with an NP machine N' to form an $\mathrm{NPF}//\mathrm{PF}^{\mathrm{SAT}[2r(n)]}$ computation. On input x, D calculates the census $c_1, \ldots, c_{r(n)}$ and passes this information to N'. The NP machine N' will simulate the original machine N level by level. At each level i, N' guesses c_i formulas, checks that they are satisfiable and verifies that they are actually queried by N on level i. Only the computation paths of N' that manage to find c_i satisfiable level i queries will continue the simulation of $N(x)$. These paths have the list of all satisfiable level i queries and know that any other level i query is unsatisfiable. Thus, N' can complete the simulation of $N(x)$ on all computation paths and will output exactly the values that $N(x)$ outputs. □

The proof of the preceding lemma can also be used to prove the analogous statement for language classes. The following lemma is not a direct corollary of Lemma 10 because the characteristic functions of languages recognized by $\mathrm{NP}^{\mathrm{SAT}[r(n)]}$ machines are not necessarily computable by $\mathrm{NPF}^{\mathrm{SAT}[r(n)]}$ machines. This is because $\mathrm{NPF}^{\mathrm{SAT}[r(n)]}$ functions must be total, hence must output 0 when the $\mathrm{NP}^{\mathrm{SAT}[r(n)]}$ machine *rejects*.

Lemma 11 Let $r(n) \in n^{O(1)}$ be a polynomial-time computable function, then

$$\mathrm{NP}//\mathrm{PF}^{\mathrm{SAT}[r(n)]} \subseteq \mathrm{NP}^{\mathrm{SAT}[r(n)]} \subseteq \mathrm{NP}//\mathrm{PF}^{\mathrm{SAT}[2r(n)]}.$$

Lemmas 10 and 11 show that by doubling the number of queries to SAT, we can convert an $\mathrm{NPF}^{\mathrm{SAT}[r(n)]}$ machine which asks its queries "on-the-fly" into an equivalent $\mathrm{NPF}//\mathrm{PF}^{\mathrm{SAT}[2r(n)]}$ machine which asks its queries deterministically. In the next lemma, we show that if $r(n)$ is logarithmically bounded, then we do not need the extra $r(n)$ queries for this conversion.

Lemma 12 Let $r(n) \in O(\log n)$ be a polynomial-time computable function, then

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

Proof: The containment $\mathrm{NPF}//\mathrm{PF}^{\mathrm{SAT}[r(n)]} \subseteq \mathrm{NPF}^{\mathrm{SAT}[r(n)]}$ is trivial, so we only need to show that $\mathrm{NPF}^{\mathrm{SAT}[r(n)]} \subseteq \mathrm{NPF}//\mathrm{PF}^{\mathrm{SAT}[r(n)]}$. This is accomplished using the mind-change technique. This same technique was used by Beigel [Bei91] to show that

$$\mathrm{P}^{\mathrm{SAT}||[2^k-1]} = \mathrm{P}^{\mathrm{SAT}[k]}.$$

Using the census trick, Hemachandra [Hem89] was only able to show that

$$\mathrm{P}^{\mathrm{SAT}[k]} \subseteq \mathrm{P}^{\mathrm{SAT}\|[2^k-1]} \subseteq \mathrm{P}^{\mathrm{SAT}[k+1]}.$$

The mind change proof was necessary to remove the one extra query. Our current situation is analogous, except here we remove $r(n)$ extra queries.

Let N be an $\mathrm{NPF}^{\mathrm{SAT}[r(n)]}$ machine. Consider the full computation tree of N on some input x. This computation tree is a combination of a nondeterministic computation tree and an oracle query tree. There are two types of branching nodes in this computation tree: nondeterministic nodes and oracle query nodes. At a nondeterministic node, the machine N chooses one of the succeeding paths nondeterministically. At an oracle query node, the machine N takes one of two branches depending on the answer to the oracle query. The YES branch represents the computation of $N(x)$ after the oracle query assuming that the oracle answered YES. The NO branch represents the computation of $N(x)$ assuming the oracle answered NO. Of course, only one of the two branches is correct with respect to the oracle SAT. For this proof, we need to consider paths in the computation tree where the oracle queries are not answered correctly. Note that the behavior of $N(x)$ on an incorrect branch is not guaranteed. In particular, on an incorrect branch, $N(x)$ may ask more than $r(n)$ oracle queries. This is one reason why the census trick does not work for this proof.

The size of the full computation tree is exponential. However, we only need to consider paths where oracle queries occur. Hence, we define a subtree T to be a *plausible subtree* of the computation tree if the following conditions are satisfied

1. The root of the subtree is the initial configuration of $N(x)$.
2. The number of query nodes in T is $\leq r(n)$.
3. For each query node Q in T, only one of its succeeding YES and NO branches is in T. If the YES branch is taken, we call Q a positive query node. Otherwise, we call Q negative.
4. If Q is a positive query node in T, then Q is satisfiable (i.e., if the query node Q asks the question "$F \in \mathrm{SAT}$?", then $F \in \mathrm{SAT}$).

Note that a satisfiable query node can be a positive query node or a negative query node in a plausible subtree, but an unsatisfiable query node must be negative. Also, the leaves in a plausible subtree need not be the leaves of the full computation tree. That is, a path from the root to a leaf of a plausible subtree may represent only the first steps of a possible computation of $N(x)$. Furthermore, an NP machine can recognize whether a subtree T is a plausible subtree of $N(x)$, because it can check whether $F \in \mathrm{SAT}$ for each positive query node. In addition, since $r(n)$ is polynomially bounded, there exists a plausible subtree T_c of polynomial size such that every query asked by $N^{\mathrm{SAT}}(x)$ (using the SAT oracle) is a query node in T_c and each branch in T_c following a query node is the correct branch with respect to the SAT oracle.

We now define a partial ordering $\prec$ on the set of plausible subtrees. The goal of this definition is to make T_c a maximal element in the ordering and to guarantee that the longest chain in the ordering contains no more than $2^{r(n)}$ plausible trees. Intuitively, we want to define $\prec$ so that $T_1 \prec T_2$ if T_2 has more positive query nodes than T_1. The idea is that since every satisfiable query in T_c is positive, T_c would contain the "most" positive query nodes. However, we need to take into account the query nodes in a plausible tree on a NO branch after a satisfiable query node. Since the NO branch is the wrong branch, the queries made in the NO branch are not included in T_c. These "false" positive query nodes must be excluded somehow. Thus, for two plausible subtrees T_1 and T_2, we say that the transition from T_1 to T_2 constitutes a mind change, written $T_1 \prec T_2$, if one of the following conditions holds.

Mind Change 1: $T_1 \subseteq T_2$ — i.e., every path in T_1 is a path in T_2 — and T_2 contains a positive query node that is not in T_1.

Mind Change 2: At least one query node makes a *mind change* from T_1 to T_2. That is, there exists a query node Q that is present in both T_1 and T_2 such that the path from the root to Q is identical in T_1 and T_2, Q is a negative query node in T_1 and a positive query node in T_2. Furthermore, every path in the computation tree of $N(x)$ that does not include a query node making a mind change is either in both T_1 and T_2 or in neither.

Under the mind change ordering $\prec$, the plausible subtree T_c described above is maximal. If $T_c \prec T$ for some plausible subtree T under Mind Change 1, then T contains at least one query made by $N^{\mathrm{SAT}}(x)$ that is not in T_c. This contradicts the assumption that T_c includes every query made by $N^{\mathrm{SAT}}(x)$. If $T_c \prec T$ under Mind Change 2, then the query node which makes the mind change must be satisfiable and be a negative query node in T_c. This contradicts the assumption that T_c takes the correct branch after every query.

Now, let $T_{\max}$ be a maximal plausible subtree in the $\prec$ ordering. We prove the following claims about $T_{\max}$.

CLAIM 1: Every satisfiable query node in $T_{\max}$ must be a positive query node.

PROOF OF CLAIM 1: Suppose that $T_{\max}$ takes the NO branch after some satisfiable query node Q. Then, we can define T' to be $T_{\max}$ modified such that all the paths following Q are replaced by a single path of $N(x)$ taking the YES branch after Q. This new path terminates right after the query node Q. Thus the number of query nodes in T' is no more than the number of query nodes in $T_{\max}$. Since Q is satisfiable, T' is a plausible subtree and $T_{\max} \prec T'$. This contradicts the maximality of $T_{\max}$. Therefore, $T_{\max}$ must take the YES branch after every satisfiable query.

CLAIM 2: Every query node in $T_{\max}$ is also a query node in T_c.

PROOF OF CLAIM 2: Since T_c includes every query actually asked by $N^{\mathrm{SAT}}(x)$, the only query nodes of the full computation tree that are not present in T_c are those that follow a wrong branch in some previous query. By the definition of plausible subtree, every YES branch taken by $T_{\max}$ is correct. By Claim 1, every NO branch taken by $T_{\max}$ is also correct. Hence, every query node in $T_{\max}$ is a query that $N^{\mathrm{SAT}}(x)$ makes (using the correct oracle).

CLAIM 3: $T_{\max}$ includes every satisfiable query in T_c.

PROOF OF CLAIM 3: Suppose that there are satisfiable queries in T_c that are not in $T_{\max}$. Then, let Q be such a query node which appears after the smallest number of time steps. (I.e., every query on the path from the root to Q is either unsatisfiable, or satisfiable and appears in $T_{\max}$.) Let p_y be a computation path in T_c which passes through Q. Since Q is satisfiable, p_y takes the YES branch after Q. Let T' be the tree produced by grafting p_y onto $T_{\max}$. By Claim 2, T' contains only query nodes in T_c. Thus the number of query nodes in T' is still bounded by $r(n)$. Furthermore, since $T_{\max}$ did not include Q, T' takes only the YES branch the query node Q. Therefore, T' is a plausible subtree and $T_{\max} \prec T'$. This violates the maximality of $T_{\max}$. Thus, $T_{\max}$ must include every satisfiable query in T_c.

CLAIM 4: The maximum number of mind changes is bounded by $2^{r(n)} - 1$.

PROOF OF CLAIM 4: We prove this claim by induction on the number of queries $r(n)$. First, consider the base case where $r(n) = 1$. Let T_0 be a plausible subtree that contains no query nodes, let T_1 be a plausible subtree with a single negative query node and let T_2 be a plausible subtree

with a single positive query node. The trees T_1 and T_2 might not actually exist, since $N(x)$ might not ask any queries or it might ask an unsatisfiable query. Nevertheless, if T_1 and/or T_2 exists, then $T_0 \prec T_2$ and $T_1 \prec T_2$. However, T_0 followed by T_1 does not constitute a mind change, since T_1 does not contain any positive query nodes. Thus, when $r(n) = 1$, the maximum number of mind changes is 1.

For the induction case, consider a plausible subtrees: $T_0 \prec T_1 \prec T_2 \prec \cdots \prec T_m$. Let Q be the first query node on some computation path of T_0. Note that the path from the root to Q must appear in every T_i by the definition of $\prec$. Let j be smallest such that Q is a positive query node in T_j. If no such T_j exists (say if Q is unsatisfiable), let $j = m+1$. Then by the definition of $\prec$, for all $i > j$, Q must also be a positive query node in T_i. For $0 \leq i < j$, let T'_i be the plausible subtree T_i modified by replacing the query node Q by its subtree in the NO branch. For $j \leq i \leq m$, let T'_i be T_i where Q is replaced by the subtree in its YES branch. (Since this is strictly a combinatorial proof, we need not be concerned that the sequence of plausible subtrees corresponds to the computation of any Turing machine.) The sequences $T'_0 \prec \cdots T'_{j-1}$ and $T'_j \prec \cdots \prec T'_m$ are two sequences of plausible subtrees using at most $r(n) - 1$ queries. (The second sequence may be empty.) Each of the two sequences can make at most $2^{r(n)-1} - 1$ mind changes. The transition from T_{j-1} to T_j counts as an additional mind change. Hence the total number of mind changes is bounded by

$$2^{r(n)-1} - 1 + 1 + 2^{r(n)-1} - 1 = 2^{r(n)} - 1,$$

and we have proven Claim 4.

An NP oracle can answer the question:

Does there exist a chain of plausible subtrees which makes at least k mind changes?

The question is an NP question because we only need to consider computation paths that involve oracle queries. Hence the size of the plausible subtrees that we need to consider are polynomially bounded. Furthermore, since $r(n) \in O(\log n)$, Claim 4 guarantees that the maximum number of mind changes is polynomially bounded. Thus, an NP machine can guess $k+1$ subtrees $T_0, T_1, \ldots T_k$ of the full computation tree, verify that they are indeed plausible subtrees and check that $T_0 \prec T_1 \prec \cdots \prec T_k$. Therefore, a $\mathrm{PF}^{\mathrm{SAT}[r(n)]}$ machine can determine the largest number of mind changes using binary search.

Finally, we show how an $\mathrm{NPF}//\mathrm{PF}^{\mathrm{SAT}[r(n)]}$ computation can simulate the $\mathrm{NPF}^{\mathrm{SAT}[r(n)]}$ computation $N(x)$. The $\mathrm{PF}^{\mathrm{SAT}[r(n)]}$ phase computes m, the maximum number of mind changes (as described above) and passes m to an NP machine N'. The machine N' then guesses $m+1$ plausible subtrees $T_0, \ldots, T_m$ and checks that $T_0 \prec \cdots \prec T_m$. By Claim 3, every satisfiable query in T_c is a positive query in T_m. Thus, N' can obtain a list of all satisfiable queries made by $N^{\mathrm{SAT}}(x)$. Hence, N' can simulate $N^{\mathrm{SAT}}(x)$ step by step. When $N^{\mathrm{SAT}}(x)$ queries SAT, N' simply checks if this query is on the list of all satisfiable queries. □

The following lemma shows the equivalence between parallel (or truth-table) queries and serial (or adaptive) queries for nondeterministic bounded query classes. For *deterministic* bounded query classes, the relationship between parallel and serial queries is a feature that distinguishes bounded query language classes from bounded query function classes. For example, in the case of language classes, Beigel [Bei91] showed that for all constants k, $\mathrm{P}^{\mathrm{SAT}[k]} = \mathrm{P}_{\mathrm{tt}}^{\mathrm{SAT}[2^k-1]}$. In contrast, for function classes, Amir, Beigel and Gasarch [ABG90] showed that $\mathrm{PF}^{\mathrm{SAT}[k]}$ does not contain $\mathrm{PF}_{\mathrm{tt}}^{\mathrm{SAT}[k+1]}$ unless PH collapses. In the following lemma, the trade-off between parallel and serial queries for nondeterministic bounded query classes is the same for both language and function classes.

Thus, the structure of the nondeterministic bounded query classes is similar to the structure of the deterministic bounded query *language* classes.

Lemma 13 Let $r(n) \in O(\log n)$ be a polynomial-time computable function, then

1. $\mathrm{NPF}//\mathrm{PF}_{\mathrm{tt}}^{\mathrm{SAT}[2^{r(n)}-1]} = \mathrm{NPF}//\mathrm{PF}^{\mathrm{SAT}[r(n)]} = \mathrm{NPF}^{\mathrm{SAT}[r(n)]}$.
2. $\mathrm{NP}//\mathrm{PF}_{\mathrm{tt}}^{\mathrm{SAT}[2^{r(n)}-1]} = \mathrm{NP}//\mathrm{PF}^{\mathrm{SAT}[r(n)]} = \mathrm{NP}^{\mathrm{SAT}[r(n)]}$.

Proof: By Lemma 12, $\mathrm{NPF}^{\mathrm{SAT}[r(n)]} = \mathrm{NPF}//\mathrm{PF}^{\mathrm{SAT}[r(n)]}$. Now consider the $\mathrm{PF}^{\mathrm{SAT}[r(n)]}$ phase of the $\mathrm{NPF}//\mathrm{PF}^{\mathrm{SAT}[r(n)]}$ computation. There are at most $2^{r(n)}-1$ queries in the entire oracle query tree of this computation. Moreover, these queries can be generated in deterministic polynomial time and asked in parallel. Thus, $\mathrm{PF}^{\mathrm{SAT}[r(n)]} \subseteq \mathrm{PF}_{\mathrm{tt}}^{\mathrm{SAT}[2^{r(n)}-1]}$ and $\mathrm{NPF}^{\mathrm{SAT}[r(n)]} \subseteq \mathrm{NPF}//\mathrm{PF}_{\mathrm{tt}}^{\mathrm{SAT}[2^{r(n)}-1]}$.

To prove that $\mathrm{NPF}//\mathrm{PF}_{\mathrm{tt}}^{\mathrm{SAT}[2^{r(n)}-1]} \subseteq \mathrm{NPF}^{\mathrm{SAT}[r(n)]}$, we program an $\mathrm{NPF}^{\mathrm{SAT}[r(n)]}$ machine to use $r(n)$ queries to SAT and binary search to determine the number of satisfiable formulas among the queries asked by the $\mathrm{PF}_{\mathrm{tt}}^{\mathrm{SAT}[2^{r(n)}-1]}$ machine. Given the census of the queries, c, the NPF machine can determine which queries are satisfiable by guessing the c queries and their satisfying assignments. Then, the NPF machine has the answers to all the queries made by the $\mathrm{NPF}//\mathrm{PF}_{\mathrm{tt}}^{\mathrm{SAT}[2^{r(n)}-1]}$ machine. So, it can carry out the simulation step by step.

In the case of language classes, we prove that $\mathrm{NP}//\mathrm{PF}_{\mathrm{tt}}^{\mathrm{SAT}[2^{r(n)}-1]} = \mathrm{NP}//\mathrm{PF}^{\mathrm{SAT}[r(n)]}$ analogously with the exception that the final simulation is for language recognition. The proof that $\mathrm{NP}//\mathrm{PF}^{\mathrm{SAT}[r(n)]} = \mathrm{NP}^{\mathrm{SAT}[r(n)]}$, is analogous to the proof of Lemma 12. □

Finally, by extending a result of Köbler and Thierauf [KT94], we can show an exact relationship between the Boolean Hierarchy and nondeterministic bounded query *language* classes.

Lemma 14 Let $r(n)$ be a monotonically increasing polynomial time computable function such that $r(n) \leq n^{\epsilon}$ for some constant $\epsilon < 1$. Then

$$\mathrm{NP}//\mathrm{PF}_{\mathrm{tt}}^{\mathrm{SAT}[r(n)]} = \mathrm{BH}_{2r(n)+1}.$$

Proof: To show that $\mathrm{BH}_{2r(n)+1} \subseteq \mathrm{NP}//\mathrm{PF}_{\mathrm{tt}}^{\mathrm{SAT}[r(n)]}$, let $L \in \mathrm{BH}_{2r(n)+1}$ and let N be an NP machine such that

$$x \in L \iff \max(\{i \mid 1 \leq i \leq 2r(n)+1 \text{ and } N(x,i) \text{ accepts}\} \cup \{0\}) \text{ is odd}.$$

Let t be the largest i such that i is even, $1 \leq i \leq 2r(n)$ and $N(x,i)$ accepts. The value of t can be determined using $r(n)$ parallel queries to the SAT oracle in the $\mathrm{PF}_{\mathrm{tt}}^{\mathrm{SAT}[r(n)]}$ phase of the computation. Note that $x \in L$ if and only if $N(x, t+1)$ accepts. Thus, the $\mathrm{PF}_{\mathrm{tt}}^{\mathrm{SAT}[r(n)]}$ machine will simply compute t and pass t to the NP machine of the nondeterministic phase which checks if $N(x, t+1)$ accepts. Therefore, $\mathrm{BH}_{2r(n)+1} \subseteq \mathrm{NP}//\mathrm{PF}_{\mathrm{tt}}^{\mathrm{SAT}[r(n)]}$.

Conversely, suppose that $L \in \mathrm{NP}//\mathrm{PF}_{\mathrm{tt}}^{\mathrm{SAT}[r(n)]}$ via an NP machine N and a deterministic polynomial time machine D. We construct an NP machine N' to prove that $L \in \mathrm{BH}_{2r(n)+1}$. For a given input x, let Q be the set of $r(n)$ parallel queries to SAT asked by D on input x. On input $(x, 2i)$, N' nondeterministically chooses a subset $Q' \subseteq Q$ with i strings and verifies that every string in Q' is satisfiable. If such a set Q' is found, N' accepts $(x, 2i)$. On input $(x, 2i+1)$, N' again chooses a subset $Q' \subseteq Q$ with i strings and verifies that $Q' \subseteq \mathrm{SAT}$. Then, N' simulates the computation of $D(x)$ assuming that every query in Q' is satisfiable and every query in $Q - Q'$ is unsatisfiable.

Finally, using the output from $D(x)$, N' simulates the computation of N on input $(x, D(x))$ and accepts if and only if N does.

We claim that $x \in L$ if and only if $t = \max(\{i \mid N'(x,i) \text{ accepts}\} \cup \{0\})$ is odd. To see this, let $z = \|Q \cap \mathrm{SAT}\|$. Then, $N'(x, 2z)$ will accept and for all $i \geq 2z+2$, $N'(x,i)$ will reject. Thus, t is either $2z$ or $2z+1$. If $x \in L$, then some computation path of $N'(x, 2z+1)$ will accept. If $x \notin L$, then $N'(x, 2z+1)$ must reject since every computation path of $N(x, D(x))$ rejects. Thus, $L \in \mathrm{BH}_{2r(n)+1}$. □

The relationship between nondeterministic bounded query *function* classes and the Boolean Hierarchy is also very tight. The following lemma states that if the function hierarchy collapses, then so does the Boolean Hierarchy. Recall that if the Boolean Hierarchy collapses, then so does PH [Kad88, BCO93, CK96, Wag88, Wag90]. Thus, nondeterministic bounded query hierarchy does not collapse unless PH collapses.

Lemma 15 Let $f(n)$ be a polynomial-time computable function such that for some $\epsilon < 1$, $f(n) \leq \epsilon \log n$. Then, for $r(n) = 2^{f(n)+1} - 1$,

$$\mathrm{NPF}^{\mathrm{SAT}[f(n)]} = \mathrm{NPF}^{\mathrm{SAT}[f(n)+1]} \Longrightarrow \mathrm{BH}_{r(n)} = \mathrm{coBH}_{r(n)}.$$

Proof: Let $s(n) = 2^{f(n)} - 1$. We show that the Boolean Hierarchy collapses at level $r(n) = 2s(n)+1$ under these assumptions. Let $A = \mathrm{BL}_{r(n)}$. By Lemmas 12 and 13,

$$A \in \mathrm{NP}/\!/\mathrm{PF}_{\mathrm{tt}}^{\mathrm{SAT}[s(n)]} = \mathrm{NP}/\!/\mathrm{PF}^{\mathrm{SAT}[f(n)]}.$$

Thus, the characteristic function of A, χ^A, can be computed in $\mathrm{PF}^{\mathrm{SAT}[f(n)+1]}$ (simply ask the same $f(n)$ queries as the $\mathrm{NP}/\!/\mathrm{PF}^{\mathrm{SAT}[f(n)]}$ machine followed by one query to SAT to see if the NP machine accepts). Then, the characteristic function of $\overline{A}$, $\chi^{\overline{A}}$, is also computable in $\mathrm{PF}^{\mathrm{SAT}[f(n)+1]}$. However,

$$\mathrm{PF}^{\mathrm{SAT}[f(n)+1]} \subseteq \mathrm{NPF}^{\mathrm{SAT}[f(n)+1]} \subseteq \mathrm{NPF}^{\mathrm{SAT}[f(n)]},$$

so $\chi^{\overline{A}} \in \mathrm{NPF}^{\mathrm{SAT}[f(n)]}$. Then, $\overline{A}$ can be recognized in $\mathrm{NP}^{\mathrm{SAT}[f(n)]} = \mathrm{NP}/\!/\mathrm{PF}_{\mathrm{tt}}^{\mathrm{SAT}[s(n)]}$. Since $\mathrm{NP}/\!/\mathrm{PF}_{\mathrm{tt}}^{\mathrm{SAT}[s(n)]} = \mathrm{BH}_{r(n)}$, $\overline{A} = \mathrm{coBL}_{r(n)} \in \mathrm{BH}_{r(n)}$. Thus, $\mathrm{BH}_{r(n)} = \mathrm{coBH}_{r(n)}$. □

Corollary 16 Let $f(n)$ be a polynomial-time computable function such that for some $\epsilon < 1$, $f(n) \leq \epsilon \log n$. Then, $\mathrm{NPF}^{\mathrm{SAT}[f(n)]} = \mathrm{NPF}^{\mathrm{SAT}[f(n)+1]}$ implies $\mathrm{PH} \subseteq \Sigma_3^{\mathrm{P}}$.

4 Bounded queries and approximation

In this section, we show that certain NP-approximation problems are complete problems for the nondeterministic bounded query classes. The theorems in this section draw together the results and proof techniques from several sources in the literature. Chang, Gasarch and Lund [CG93, CGL97, Cha96] provide detailed calculations of the trade-off between the number of oracle queries and the closeness of the approximation. However, these results only deal with the cost of the approximate solution (e.g., CHROMATIC NUMBER rather than GRAPH COLORING). Using the proofs of Khanna *et al.* [KMSV94], these calculations can be extended to work with witness preserving reductions. Additional results connecting bounded queries and approximations and on witness preserving reductions can be found in the works of Crescenzi *et al* [CKST95, CT94].

The upper bounds on the complexity of finding approximate solutions to NP-optimization problems are easy to establish. For example, an $\mathrm{NPF}^{\mathrm{SAT}[\log n]}$ can solve MAXCLIQUE by first using its queries to SAT and binary search to find the size of the largest clique in the graph. Then, the machine uses nondeterminism to guess and verify the vertices which belong to a largest clique. Since there may be more than one maximum clique, the nondeterministic machine may output more than one value. Similarly, using $\log\log n$ queries to SAT, an NP machine can find a 2-approximate clique. First, the machine uses binary search to determine which of the intervals $[1,2],[2,4],[4,8],\ldots$ contains the size of the largest clique. Since we only need to consider $\lceil \log n \rceil$ intervals, $\lceil \log \lceil \log n \rceil \rceil$ queries to SAT is sufficient for the binary search. Suppose that $[x, 2x]$ is that interval, then the machine nondeterministically chooses $y \in [x, 2x]$, guesses y vertices in the graph and verifies that these vertices form a clique. This technique extends easily to non-constant approximation factors $k(n)$ that are polynomial time computable. Furthermore, if there exists a polynomial time algorithm that can find approximate solutions within a factor of $k'(n)$, then we can use this approximation algorithm to reduce the number of intervals we have to consider. Hence, fewer queries to SAT would be needed in the binary search.

Lemma 17

- For polynomial time computable $k(n)$, with $1 \leq k(n) \leq n$, there exists a function in $\mathrm{NPF}^{\mathrm{SAT}[q(n)]}$ which $k(n)$-approximates MAXCLIQUE, where $q(n) = \left\lceil \log \left\lceil \log_{k(n)} n \right\rceil \right\rceil$.
- For polynomial time computable $k(n)$, with $1 \leq k(n) \leq 2^n$, there exists a function in $\mathrm{NPF}^{\mathrm{SAT}[q(n)]}$ which $k(n)$-approximates TSP, where $q(n) = \lceil \log \lceil n / \log k(n) \rceil \rceil$.
- 2-approximating SET COVER can be solved in $\mathrm{NPF}^{\mathrm{SAT}[\log\log\log n + O(1)]}$.

Proof: This lemma follows immediately from the preceding discussion. SET COVER can be approximated within a factor of $O(\log n)$ [Lov75]. Also, by the convention stated in Definition 8, the length of TSP tour cannot exceed 2^n where n is the number of vertices in the graph. □

4.1 Completeness of approximating MAXCLIQUE

It is fairly straightforward to establish the upper bounds for NP-approximation problems. However, it is more difficult to show that these NP-approximation problems are in fact hard problems for the nondeterministic bounded query classes. For our results on MAXCLIQUE, we need the following lemma from probabilistically checkable proofs. This lemma was communicated to the author by Madhu Sudan, but we are not aware of a written proof of a lemma in this exact form. So, we sketch

the proof below. Some familiarity with probabilistically checkable proofs and the deterministic construction of disperser graphs is necessary. For this, we refer there reader to the literature [FGL+91, AS92, ALM+92, Aro94, CW89, CW]

Lemma 18 There exist integer constants $0 < s < b < d$ and a polynomial time computable function h such that given a 3CNF formula F with t variables, $h(F)$ constructs an undirected graph G with t^d vertices where:

1. $F \in \text{SAT} \Longrightarrow \omega(G) = t^b$.
2. $F \notin \text{SAT} \Longrightarrow \omega(G) = t^s$.
3. Given the vertices of a clique in G with $t^s + 1$ vertices, we can construct a clique of G with t^b vertices and a satisfying assignment for F in polynomial time.

Proof: We start with a probabilistically checkable proof for SAT where the verifier uses $O(\log n)$ random bits and $O(1)$ proof bits to achieve a probabilistic error of $1/2$ in verifying the proof [AS92, ALM+92, Aro94]. Specifically, we assume that given a formula F with t variables as input, the verifier uses exactly $c_1 \log t$ random bits on each computation path and looks at exactly c_2 bits of the proof. Moreover, the locations of the proof bits are determined only by the choice of random bits (i.e., the locations are not adaptively dependent on the settings of other proof bits). If $F \in \text{SAT}$, then there exists a proof which causes the verifier to accept on all of the t^{c_1} random paths. On the other hand, if the $F \notin \text{SAT}$, then given any proof, the verifier accepts on at most half of the random paths.

The computation tree of the verifier can be transformed into a graph in the standard way [FGL+91]. Here each path in the computation tree is specified by the $c_1 \log t$ random bits and by the settings of the c_2 proof bits used by the verifier. Thus, there are $2^{c_2} t^{c_1}$ paths in the computation tree. Two computation paths are *inconsistent* if they assume different settings for some proof bit that is used by both paths. (Note that two paths using the same sequence of random bits must be inconsistent, since they assume different settings for the same proof bits.) Let G' be a graph with $2^{c_2} t^{c_1}$ vertices, each of which corresponds to a path in the computation tree. Every pair of vertices in G' is connected by an edge unless the vertices correspond to inconsistent paths. Using this construction,

$$\begin{aligned} F \in \text{SAT} &\Longrightarrow \omega(G') = t^{c_1} \\ F \notin \text{SAT} &\Longrightarrow \omega(G') \leq t^{c_1}/2. \end{aligned}$$

Now, suppose that we are given the vertices of a clique in G' with $t^{c_1}/2 + 1$ vertices. From above, we know that there exists a clique with t^{c_1} vertices, but it may be difficult to find such a clique. We first reconstruct a proof that causes the verifier to accept on the paths indicated by the vertices of the given clique. To do this, we use the settings of the proof bits assumed by each of these paths. However, some bits of the proof may not have been queried by any of these paths, so we only have a partial proof. It turns out that since the proof was encoded using a linear code, we can use the error-correction algorithm on the partial proof to obtain a full proof. Once the full proof is obtained we can produce the set paths that are consistent with the proof. This gives us the vertices of a t^{c_1}-clique in G'. This feature of probabilistically checkable proofs was used in Theorem 4 of [KMSV94] to obtain a complete satisfying assignment of a specially constructed formula given an assignment that only satisfies a large portion of the clauses.

We make the additional note that the full proof is actually an encoding of a satisfying assignment for F. This may not be the case if the verifier accepts some "bogus" proofs in addition to the correct

proof when $F \in$ SAT. However, the construction the verifier guarantees that even when $F \in$ SAT, the verifier will accept with probability $< 1/2$ given a proof that is not an encoding of a satisfying assignment for F (q.v. the definition of "normal form verifier" [Aro94, Definition 3.2 pp. 19–20].) Thus, by inverting the encoding, we can also obtain a satisfying assignment for F.

The graph G' constructed above has an approximation "gap" of only 2, since the value of $\omega(G')$ changes by a factor of 2 between the cases where $F \in$ SAT and $F \notin$ SAT. This "gap" can be amplified by reducing the error probability of the verifier using explicit construction of disperser graphs [CW89, CW].[4] The disperser graphs allow us to construct a verifier that uses $b \log t$ random bits and $c_3 \log t$ proof bits such that the probability that the verifier mistakenly accepts a "bogus" proof is $\leq t^s/t^b$. Each path of the new verifier accomplishes this by simulating the old verifier on $c_4 \log t$ paths. However, since the paths of the old verifier are chosen as prescribed by the disperser, only $b \log t$ random bits are used instead of $c_1 c_4 \log^2 t$ random bits. On the other hand, the verifier does use $c_2 c_4 \log t$ proof bits. The new verifier accepts if and only if the old verifier accepted in all $c_4 \log t$ simulations.

The construction of the disperser guarantees that given any set X of $\leq t^{c_1}/2$ paths of the old verifier, at most t^s random paths of the new verifier will simulate only paths from X. Now, suppose that $F \notin$ SAT. Let X be the set of accepting paths of the old verifier. Since $F \notin$ SAT, $|X| \leq t^{c_1}/2$. Let Y be the set of paths in the new verifier that simulate only paths from X. By the disperser graph property, $|Y| \leq t^s$. The new verifier will accept on paths in Y. However, on any path outside Y, the new verifier must simulate a rejecting path of the old verifier and reject. Thus, the new verifier will accept with probability at most t^s/t^b. On the other hand, suppose that $F \in$ SAT. Then, there exists a proof which makes the new verifier accept on every path, since the old verifier accepts this proof on every path.

Finally, construct the graph G from the computation tree of the new verifier in the same manner that G' was constructed from the old verifier. Clearly, if $F \notin$ SAT then $\omega(G) = t^s$.[5] If $F \in$ SAT, then $\omega(G) = t^b$. Furthermore, suppose that we are given a clique with $t^s + 1$ vertices of G. These vertices correspond to a set Y of $t^s + 1$ random paths in the new verifier. Let X be the set of random paths in the old verifier that are simulated by these $t^s + 1$ paths. We know that the paths in X are accepting paths, but we worry that there are too few paths in X to recover a complete proof that $F \in$ SAT. So, suppose that $|X| \leq t^{c_1}/2$. By properties of the disperser graph, $|Y|$ must be at most t^s which is a contradiction. Thus, $|X|$ must be at least $t^{c_1}/2 + 1$. Then, by the previous discussion on G', we can recover a complete proof which makes both verifiers accept with probability 1. Thus, we can recover a clique in G with t^b vertices and a satisfying assignment for F. □

For the graph G constructed in Lemma 18, no polynomial time algorithm can find an n^ϵ-approximate clique of G unless P = NP for $\epsilon = (b - s)/d$. In the rest of this section, the constants b, s, d and ϵ refer to these constants. Using Lemma 18, we can construct a graph whose $k(n)$-approximate cliques gives us enough information to reconstruct a satisfying assignment of a 3CNF formula from a sequence of formulas. The proof of the lemma below is a modification of the Construction Lemma from [Cha96]. These modifications take advantage of stronger results in Lemma 18.

[4] In particular, in the terminology of Cohen and Wigderson, we are using Theorem 4.8 of [CW] with $\alpha = 1/2$, $l = c_4 \log t$ and $N = t^{c_1}$ to obtain a $(1/2, 1/\text{poly}, 1)$ disperser. One can also increase the gap in the clique size using booster graphs [Aro94] or product graphs [AFWZ95]. However, these results only provide estimates on $\omega(G)$. Our proofs require that we know possible values of $\omega(G)$ exactly.

[5] W.l.o.g. we can add a t^s clique of new vertices to G in order to guarantee that $\omega(G)$ is not less than t^s.

Lemma 19 Let $k(n)$ be a polynomial time computable function such that $k(n) \geq 1 + \sqrt{2}/n^{\delta}$ for $\delta = \epsilon/(4+4\epsilon)$, where $\epsilon = (b-s)/d$. Let $m = t^{b-s+d}$ and let $F_0, \ldots, F_{r(t)-1}$ be a sequence of $r(t)$ 3CNF formulas each with t variables such that $r(t) \leq \log_{k(m)} t^{(b-s)/2}$. Define

$$z = \max\{i \mid 1 \leq i \leq r(t) - 1 \text{ and } F_i \in \text{SAT}\}.$$

Then, we can construct in polynomial time a graph H with m vertices such that given the vertices of any $k(m)$-approximate clique of H, we can in polynomial time determine the value of z and construct a satisfying assignment of F_z.

Proof: The main difficulty in this construction is that the approximation factor $k(m)$ is dependent on the size of the graph H that we construct. However, we need to use the value of $k(m)$ to construct H in the first place. To break this circular dependency, we will first construct a graph H' with fewer than m vertices. Then the graph H is produced from H' by simply adding $m - |H'|$ unconnected vertices. Also, for notational convenience, we let $g = k(m)$ and $r = r(t)$.

To construct H', we take each F_i and produce a graph G_i with t^d vertices according to Lemma 18. Define the values $a_0, a_1, \ldots, a_{r-1}$ recursively as:

$$a_i = \begin{cases} 1 & \text{if } i = 0 \\ g \cdot a_{i-1} + 1 & \text{for } 1 \leq i \leq r-1 \end{cases}$$

We combine $\lfloor a_i \rfloor$ copies of each G_i into a graph G'_i such that $\omega(G'_i) = \lfloor a_i \rfloor \cdot \omega(G_i)$. This can be accomplished easily by connecting each vertex in every copy of G_i to every vertex in a different copy of G_i. The graph H' is simply the disjoint union of $G'_0, \ldots, G'_{r-1}$ without any additional edges.

The graph H' has $\sum_{i=0}^{r-1} \lfloor a_i \rfloor t^d$ vertices. We need an upper bound on H' in order to justify the claim that H' has $\leq m$ vertices. Since

$$a_i = \sum_{j=0}^{i} g^j = \frac{g^{i+1} - 1}{g - 1},$$

$|H'|$ is bounded by

$$\sum_{i=0}^{r-1} a_i t^d \;<\; \frac{g(g^r - 1)}{(g-1)^2} \cdot t^d \;\leq\; \frac{g}{(g-1)^2} \cdot t^{(b-s)/2} \cdot t^d.$$

Here we use the restriction on r from the hypothesis of this lemma to show that $g^r - 1 < g^r \leq t^{(b-s)/2}$. In these calculations, when $g \geq 2$, we have a trivial case because $g/((g-1)^2$ is bounded by 2. For $g \leq 2$, we obtain an upper bound on $g/(g-1)^2$ using the lower bound on $g - 1 = k(m) - 1 \geq \sqrt{2}/m^{\delta}$ and the fact that $t^{b-s} = m^{\epsilon/(1+\epsilon)}$. In either case, we have $g/(g-1)^2 \leq t^{(b-s)/2}$. Hence, H' has fewer than $m = t^{b-s+d}$ vertices. Again, our final graph H is simply the disjoint union of H' with $m - |H'|$ dummy vertices.

Now we claim that the maximum cliques in H are precisely the largest cliques in G'_z. By definition, z is the index of the last satisfiable formula in the sequence $F_0, \ldots, F_{r-1}$. Since $F_z \in$ SAT, we know from Lemma 18 that $\omega(G'_z) = \lfloor a_z \rfloor t^b$. Moreover, for all $j > z$, $F_j \notin$ SAT hence $\omega(G'_j) = \lfloor a_j \rfloor t^s$. Since the restrictions on $r(t)$ and $k(m)$ guarantee that $k(m) a_{r-1} t^s < t^b$, we conclude that the largest clique in H cannot come from G'_j for any $j > z$. In fact, the largest clique in G'_z is larger than the largest clique in G_j by at least a factor of $k(m)$. For $j < z$, the fact that $\lfloor a_z \rfloor \geq k(m) a_j$ implies that the largest cliques in G'_j is smaller than the largest clique in G'_z by a factor of $k(m)$.

Finally, note that any $k(m)$ approximate clique in H must contain more than $\lfloor a_z \rfloor t^s$ vertices from G'_z and more than t^s vertices from one of the copies of G_z. Hence, by Lemma 18 we can recover a satisfying assignment of F_z in polynomial time. □

The construction of H in Lemma 19 provides the following hardness result for approximating MaxClique. Here we simply have to convert an $\mathrm{NPF}^{\mathrm{SAT}[q(n)]}$ computation into a sequence of Boolean formulas.

Theorem 20 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 $k(\cdot)$-approximates MaxClique, then every $f \in \mathrm{NPF}^{\mathrm{SAT}[q(n)]}$ $\leq^{\mathrm{P}}_{\mathrm{m}}$-reduces to h, where $k(n) = n^{2^{-q(n)-c}}$ and $c = 1 + \log(1 + 1/\epsilon)$.

Proof: Recall that $\epsilon = (b-s)/d$, where b, s and d are the constants from Lemma 18. Let f be a total multi-valued function in $\mathrm{NPF}^{\mathrm{SAT}[q(n)]}$. Since $q(n) \in O(\log n)$, by Lemma 12 $f \in \mathrm{NPF}//\mathrm{PF}^{\mathrm{SAT}[q(n)]}$. For a fixed input string x of length n, consider the oracle query tree of the $\mathrm{PF}^{\mathrm{SAT}[q(n)]}$ phase of the $\mathrm{NPF}//\mathrm{PF}^{\mathrm{SAT}[q(n)]}$ computation for f. Since $2^{q(n)}$ is polynomially bounded, we can examine every path in the oracle query tree in polynomial time. We index the paths of the oracle query tree as described in Section 2 and let X_i be the conjunction of all the positive queries on the path with index i. Let y_i be the output of the $\mathrm{PF}^{\mathrm{SAT}[q(n)]}$ computation on the ith path of the oracle query tree. The second phase of the $\mathrm{NPF}//\mathrm{PF}^{\mathrm{SAT}[q(n)]}$ computation is performed by some NP machine N. Let N' be an NP machine which on input (x, y_i) simulates $N(x, y_i)$ and accepts on paths where N outputs a value. For each i, we take the computation of $N'(x, y_i)$ and transform it into a Boolean formula Y_i using Cook's reduction. Finally, let $F_i = X_i \wedge Y_i$. Without loss of generality, we assume that each F_i has the same number of variables t which is greater than n.

Let z be the index of the correct path in the oracle query tree. Then, X_z is satisfiable since every positive query on the correct path is really satisfiable. Furthermore, since f is a total function, $N(x, y_z)$ must output a value on some path. Hence, $F_z = X_z \wedge Y_z$ must be satisfiable. On the other hand, for all $i > z$, some positive query on the ith path is in fact unsatisfiable, so $F_i \notin \mathrm{SAT}$ for $i > z$.

Next consider approximation factors $k(n) \geq 1 + \sqrt{2}/n^{\delta}$ for some $\delta < \epsilon/(4 + 4\epsilon)$.[6] Let $r(n) = 2^{q(n)}$. By definition of $k(n)$, we have $\log \log_{k(n)} n - c = q(n)$. Since $q(n)$ is nondecreasing, it follows that $\log_{k(n)} n$ must also be nondecreasing. Let $m = t^{b-s+d}$, then $n < t < m$, so

$$r(n) \leq 2^{-c} \log_{k(n)} n \leq 2^{-c} \log_{k(m)} m.$$

From the definition of c, we have $2^{-c} = (b-s)/(2(b-s+d))$, so

$$2^{-c} \log_{k(m)} m = \frac{b-s}{2(b-s+d} \cdot \log_{k(m)} t^{b-s+d} = \log_{k(m)} t^{(b-s)/2}.$$

Thus, $r(t) \leq \log_{k(m)} t^{(b-s)/2}$ as required by Lemma 19. Therefore, we can use Lemma 19 to produce a graph H with m vertices from the formulas $F_0, \ldots, F_{r(t)-1}$.

By hypothesis h is a function which $k(\cdot)$-approximates MaxClique. So, let S be a set of vertices output by $h(H)$ that $k(m)$-approximates the largest clique in H. By Lemma 19, from S we can obtain the value of z and a satisfying assignment for F_z. This satisfying assignment also contains a satisfying assignment for Y_z. We can use this to trace an accepting path of the $N'(x, y_z)$

[6] For larger values of $k(n)$, we can use the padding argument in Section 2 to show that every function in $\mathrm{NPF}^{\mathrm{SAT}[c_1 \log n]}$ reduces to some function in $\mathrm{NPF}^{\mathrm{SAT}[c_2 \log n]}$ where $c_1 > c_2$.

computation. Since this is a path where $N(x, y_z)$ prints out a value, we can obtain an output of $f(x)$ given the vertices of a $k(m)$-approximate clique for H. Therefore, $f \leq_{\mathrm{m}}^{\mathrm{P}} h$. □

We are now in a position to prove our completeness results for approximating MaxClique. We say that 2-approximating MaxClique is complete for $\mathrm{NPF}^{\mathrm{SAT}[q(n)]}$ where $q(n) = \log\log n + O(1)$, if there is a function $h \in \mathrm{NPF}^{\mathrm{SAT}[q(n)]}$ which 2-approximates MaxClique and every function $f \in \mathrm{NPF}^{\mathrm{SAT}[q(n)]}$ $\leq_{\mathrm{m}}^{\mathrm{P}}$-reduces to h. It is not reasonable to require that every function which 2-approximates MaxClique belongs to $\mathrm{NPF}^{\mathrm{SAT}[q(n)]}$ since there are uncomputable functions which 2-approximate MaxClique.

Corollary 21

- MaxClique is $\leq_{\mathrm{m}}^{\mathrm{P}}$-complete for $\mathrm{NPF}^{\mathrm{SAT}[O(\log n)]}$.
- $(1 + 1/\log^a n)$-approximating MaxClique is $\leq_{\mathrm{m}}^{\mathrm{P}}$-complete for $\mathrm{NPF}^{\mathrm{SAT}[(a+1)\log\log n + O(1)]}$.
- 2-approximating MaxClique is $\leq_{\mathrm{m}}^{\mathrm{P}}$-complete for $\mathrm{NPF}^{\mathrm{SAT}[\log\log n + O(1)]}$.
- $(\log n)$-approximating MaxClique is $\leq_{\mathrm{m}}^{\mathrm{P}}$-complete for $\mathrm{NPF}^{\mathrm{SAT}[\log\log n - \log\log\log n + O(1)]}$.

Proof: By Lemma 17, $k(n)$-approximating MaxClique can be achieved using the stated number of queries. We give a proof for the case where $k(n) = 1 + 1/\log n$; the other cases are similar. In this case, we use the Taylor series approximation that $\ln(1 + 1/\log n) \approx 1/\log n$. Thus,

$$\log\log_{k(n)} n = \log\log n - \log\ln k(n) - \log\log e \approx \log\log n - \log(1/\log n) = 2\log\log n.$$

Each of the $\mathrm{NPF}^{\mathrm{SAT}[q(n)]}$ classes considered in this lemma is closed under $\leq_{\mathrm{m}}^{\mathrm{P}}$-reductions. So, a difference of a constant number of queries is inconsequential. Therefore, there is a function in $\mathrm{NPF}^{\mathrm{SAT}[2\log\log n + O(1)]}$ which $(1 + 1/\log n)$-approximates MaxClique.

Now, let h be any function which $(1 + 1/\log n)$-approximates MaxClique. By Theorem 20, every function in $\mathrm{NPF}^{\mathrm{SAT}[q(n)]}$ where $q(n) = \log\log_{k(n)} n - c \approx 2\log\log n - c$ reduces to h. By the padding argument in Section 2, for any constant c', every function in $\mathrm{NPF}^{\mathrm{SAT}[q(n)+c']}$ reduces to some function in $\mathrm{NPF}^{\mathrm{SAT}[q(n)]}$. Thus, every function in $\mathrm{NPF}^{\mathrm{SAT}[2\log\log n + O(1)]}$ reduces to h. □

A possible alternative definition of "completeness" is to require that the function $\mathcal{H}$ which outputs *every* 2-approximate clique belong to $\mathrm{NPF}^{\mathrm{SAT}[q(n)]}$. However, we show in Lemma 22 that $\mathcal{H}$ requires $\Omega(\log n)$ queries to SAT unless PH collapses. Intuitively, in order for $\mathcal{H}$ to output every 2-approximate clique, $\mathcal{H}$ must "know" the size of the largest clique exactly. Thus, the complexity of $\mathcal{H}$ is not a good measure of the complexity of merely 2-approximating MaxClique.

Lemma 22 Let $k(n)$ be a polynomial time computable approximation factor such that $1 \leq k(n) \leq n$. Let $\mathcal{H}$ be a multi-valued function which $k(\cdot)$-approximates MaxClique such that every $k(\cdot)$-approximate clique of an undirected graph G is an output of $\mathcal{H}(G)$. If $\mathcal{H} \in \mathrm{NPF}^{\mathrm{SAT}[q(n)]}$ for $q(n) \in o(\log n)$, then $\mathrm{PH} \subseteq \Sigma_3^{\mathrm{P}}$.

Proof: The function $\mathcal{H}$ can certainly be computed in $\mathrm{NPF}^{\mathrm{SAT}[O(\log n)]}$ — use $\log n$ queries to determine $\omega(G)$ exactly, nondeterministically guess every subset of vertices in G with at least $\lceil \omega(G)/k(n) \rceil$ vertices and output the subset of vertices if it forms a clique. Now, suppose that $\mathcal{H} \in \mathrm{NPF}^{\mathrm{SAT}[q(n)]}$ for $q(n) \in o(\log n)$. By Lemma 12, $\mathcal{H} \in \mathrm{NPF}//\mathrm{PF}^{\mathrm{SAT}[q(n)]}$ via a $\mathrm{PF}^{\mathrm{SAT}[q(n)]}$ machine D and an NP machine N. We show that MaxClique can be computed in $\mathrm{NPF}//\mathrm{PF}^{\mathrm{SAT}[q(n)]}$ as

well. We use the same $\mathrm{PF}^{\mathrm{SAT}[q(n)]}$ machine D and a new NP machine N'. Each computation path of the new machine N' simply simulates 2 computation paths of N. Suppose that N produces outputs X_1 and X_2 on both of these paths. Then, N' checks whether $|X_1| = \lceil |X_2|/k(n) \rceil$. If this is the case, then N' knows that X_2 is a maximum clique of G and that $|X_1|$ is the size of the smallest clique that still qualifies as a $k(n)$-approximate clique. Thus, N' can output X_2 as a maximum clique in G. Since N is required to output every 2-approximate clique in G, we know that some path of N' will find the suitable pair of cliques X_1 and X_2. Thus, MaxClique $\in \mathrm{NPF}^{\mathrm{SAT}[q(n)]}$. Since MaxClique is $\leq^{\mathrm{P}}_{\mathbf{m}}$-complete for $\mathrm{NPF}^{\mathrm{SAT}[O(\log n)]}$ by Corollary 21 and since $q(n) \in o(\log n)$, the Polynomial Hierarchy collapses to Σ^{P}_3 by Corollary 16 .

To see this last step, let f be a function in $\mathrm{NPF}^{\mathrm{SAT}[0.5 \log n + 1]}$. By Corollary 21, f $\leq^{\mathrm{P}}_{\mathbf{m}}$-reduces to MaxClique via polynomial time computable functions T_1 and T_2. Recall from the definition of $\leq^{\mathrm{P}}_{\mathbf{m}}$-reductions that T_1 converts an input string x of the function f into a graph G and that T_2 computes $f(x)$ using a maximum clique of G. Consider a fixed input string x of length n. By the argument above, an $\mathrm{NPF}^{\mathrm{SAT}[q(n)]}$ computation can find the largest cliques in G. Since G can have at most n^c vertices for some constant c, the $\mathrm{NPF}^{\mathrm{SAT}[q(n)]}$ computation would use at most $q(n^c)$ queries to SAT. Recall that $q(n) \in o(\log n)$. In particular, $q(n) \leq (\log n)/2c$, so $q(n^c) \leq 0.5 \log n$. Thus, the number of queries used is no more than $0.5 \log n$. Finally, by applying T_2 to any maximum clique produced by N', we can get an output of $f(x)$. So, $f \in \mathrm{NPF}^{\mathrm{SAT}[0.5 \log n]}$ and $\mathrm{NPF}^{\mathrm{SAT}[0.5 \log n + 1]} \subseteq \mathrm{NPF}^{\mathrm{SAT}[0.5 \log n]}$. Therefore, PH collapses to Σ^{P}_3 by Corollary 16. □

In the introduction of this paper, we mentioned that a $\mathrm{PF}^{\mathrm{SAT}[n^{O(1)}]}$ machine can 2-approximate MaxClique. This is accomplished using $\log \log n$ queries to approximate $\omega(G)$ within a factor of 2 then using $O(n)$ queries to identify the vertices of a 2-approximate clique. While the $\mathrm{PF}^{\mathrm{SAT}[n^{O(1)}]}$ machine model offers an advantage of not having to work with NP machines and multi-valued functions, we show in the following lemma that 2-approximating MaxClique cannot be $\leq^{\mathrm{P}}_{\mathbf{m}}$-complete for $\mathrm{PF}^{\mathrm{SAT}[n^{O(1)}]}$ unless PH collapses. In fact, 2-approximating MaxClique cannot be hard for $\mathrm{PF}^{\mathrm{SAT}[q(n)]}$ for any $q(n)$ greater than $\log \log n$ by more than a constant. Thus, resorting to a $\mathrm{PF}^{\mathrm{SAT}[n^{O(1)}]}$ machine to 2-approximate MaxClique is a waste of oracle queries.

Lemma 23 Let $q(n)$ be a polynomial time computable function such that $q(n) - \log \log n$ is asymptotically greater than any constant. If 2-approximating MaxClique is $\leq^{\mathrm{P}}_{\mathbf{m}}$-hard for $\mathrm{PF}^{\mathrm{SAT}[q(n)]}$, then PH $\subseteq \Sigma^{\mathrm{P}}_3$.

Proof: Without loss of generality, assume that $q(n) \leq 0.5 \log n$. We will show that $\mathrm{NPF}^{\mathrm{SAT}[q(n)]} \subseteq \mathrm{NPF}^{\mathrm{SAT}[q(n)-1]}$. Let f be a multi-valued function in $\mathrm{NPF}^{\mathrm{SAT}[q(n)]}$. By Lemma 12, f is computable in $\mathrm{NPF}//\mathrm{PF}^{\mathrm{SAT}[q(n)]}$ via a deterministic polynomial time machine D and an NP machine N. Fix an input string x of length n. The computation of $D(x)$ is a $\mathrm{PF}^{\mathrm{SAT}[q(n)]}$ computation. So, by assumption, $D(x)$ $\leq^{\mathrm{P}}_{\mathbf{m}}$-reduces to 2-approximating MaxClique via polynomial time computable functions T_1 and T_2. The function T_1 takes the input string x and produces a graph G that has at most n^c vertices for some constant c. A 2-approximate clique of G can be found using an $\mathrm{NPF}^{\mathrm{SAT}[\log \log n^c]}$ computation. On each path of this computation which finds a 2-approximate clique, we can modify the machine to run T_2 and obtain the output of $D(x)$. Then, these paths can simulate the computation of $N(x, D(x))$ and compute the outputs of $f(x)$. Since, $\log \log n^c < q(n) - 1$, we have shown that $f \in \mathrm{NPF}^{\mathrm{SAT}[q(n)-1]}$. Therefore, PH collapses to Σ^{P}_3 by Corollary 16. □

The proof of Lemma 23 extends to the approximation factors $\log n$ and $1 + 1/\log^a n$. However, the proof does not extend to solving MaxClique exactly. This is because we are unable to prove

that PH collapses under the assumption that $\mathrm{NPF}^{\mathrm{SAT}[\log n]} = \mathrm{NPF}^{\mathrm{SAT}[n]}$.[7] When $q(n) \leq \epsilon \log n$ for $\epsilon < 1$, we know that each additional query to SAT allows an $\mathrm{NPF}^{\mathrm{SAT}[q(n)]}$ machine to compute more functions unless PH collapses. Above this level, it remains possible that additional queries to SAT do not provide any increase in computing power. In the next section we show that the structure of the complexity classes between $\mathrm{NPF}^{\mathrm{SAT}[\log n]}$ and $\mathrm{NPF}^{\mathrm{SAT}[n]}$ is closely related to the complexity of approximating the Traveling Salesman Problem.

4.2 Completeness of approximating TSP

In this section we prove some connections between nondeterministic bounded query classes and the complexity of finding approximate solutions to the Traveling Salesman Problem. For example, we show that 2-approximating TSP is $\leq^{\mathrm{P}}_{\mathbf{m}}$-complete for $\mathrm{NPF}^{\mathrm{SAT}[O(\log n)]}$ and $(1 + n^{-\log n})$-approximating TSP is $\leq^{\mathrm{P}}_{\mathbf{m}}$-complete for $\mathrm{NPF}^{\mathrm{SAT}[O(\log^2 n)]}$. Thus, whether $O(\log^2 n)$ queries to SAT is strictly more powerful than $O(\log n)$ queries to SAT is equivalent to whether finding a $(1 + n^{-\log n})$-approximate solution to TSP is strictly harder than merely finding a 2-approximate solution. Recall that the length of a TSP tour in a graph with n vertices can be as large as 2^n. Thus, the difference between the length of an optimum TSP tour and that of a $(1 + n^{-\log n})$-approximate TSP tour can still be exponential in n. Since we are mainly concerned with query bounds above $\log n$ and since 2-approximating TSP is $\leq^{\mathrm{P}}_{\mathbf{m}}$-complete for $\mathrm{NPF}^{\mathrm{SAT}[O(\log n)]}$, we will only work with non-increasing approximation factors. While it is possible to obtain similar results for increasing approximation factors, the calculations are more involved. Hence, we omit such results from this paper.

In the following theorem, we prove the hardness of approximating TSP. In contrast to the results on MaxClique, the proofs in this section are elementary constructions and do not require theorems from probabilistically checkable proofs. The main difficulty in these proofs is that the number of paths in the oracle query tree for a $\mathrm{PF}^{\mathrm{SAT}[q(n)]}$ computation is superpolynomial when $q(n) \gg \log n$. So, in these cases we cannot look at every computation path in polynomial time.

Theorem 24 Let $f \in \mathrm{NPF}//\mathrm{PF}^{\mathrm{SAT}[q(n)]}$ where $q(n)$ is a non-decreasing polynomially bounded polynomial time computable function. Suppose that h is a multi-valued function that $k(n)$-approximates TSP where $2^{q(n)} \log k(n)$ is polynomially bounded and $k(n)$ is a non-increasing polynomial time computable function. Then, $f \leq^{\mathrm{P}}_{\mathbf{m}} h$.

Proof: Let f be a function in $\mathrm{NPF}//\mathrm{PF}^{\mathrm{SAT}[q(n)]}$ witnessed by an NP machine N and a $\mathrm{PF}^{\mathrm{SAT}[q(n)]}$ machine D. We construct an NP machine N' to simulate the $\mathrm{NPF}//\mathrm{PF}^{\mathrm{SAT}[q(n)]}$ computation. Our eventual goal is to use Karp's reduction to convert the computation of N' on an input string x of length n into a 3CNF formula F. Then we will use a standard reduction from SAT to the Hamiltonian Cycle Problem (HC) to produce a graph G [Pap94, Theorem 9.7, pp. 193–198] from the formula F. Finally, some "special" edges of G will assigned certain weights. This produces an instance of TSP such that any $k(\cdot)$-approximate tour of G will provide enough information to recover a value output by $f(x)$.

The graph G will contain no more than $p(n)$ vertices for some polynomial $p(n)$. Without loss of generality, we can assume that G has exactly $p(n)$ vertices by adding a sufficient number of dummy vertices on a single path to G. Also, without loss of generality, $p(n) \geq n$. Since $k(n)$ is non-increasing, $k(p(n)) \leq k(n)$. Thus, a $k(p(n))$-approximate TSP tour of G must also be a $k(n)$-approximate TSP tour.

[7] This is equivalent to the following perennial open problem in bounded query language classes: Does $\mathrm{P}^{\mathrm{SAT}[\log n]} = \mathrm{P}^{\mathrm{SAT}[n]}$ imply that PH collapses? This open problem is currently one of the biggest challenges in research on bounded query classes.

Now, consider the recurrence given by $a_{i+1} = k(n) \cdot a_i + 1$ where $a_0 = p(n)$. Solving the recurrence yields:

$$a_i = k(n)^i p(n) + \sum_{j=0}^{i-1} k(n)^j = k(n)^i p(n) + \frac{k(n)^i - 1}{k(n) - 1}.$$

Let $r = 2^{q(n)}$ (which may be superpolynomial). Let m denote the number of bits needed to express $\lfloor a_{r-1} \rfloor$ as a binary number. Since $2^{q(n)} \log k(n)$ is polynomially bounded, m is also polynomially bounded. Thus, a_i is polynomial time computable for all $i \leq r - 1$. Furthermore, note that:

$$k(n) \lfloor a_i \rfloor \leq k(n) a_i < \lfloor a_{i+1} \rfloor .$$

Our construction will guarantee that any TSP tour of G (not necessarily optimal) will have a weight equal to $\lfloor a_i \rfloor$ for some i, $0 \leq i \leq r - 1$. This ensures that any $k(n)$-approximate tour of G is in fact an optimal tour.

We now construct the machine N'. Consider the oracle query tree of $D(x)$ for some fixed input string x with length n. Recall that $D(x)$ is the $\mathrm{PF}^{\mathrm{SAT}[q(n)]}$ phase of the computation for $f(x)$. The oracle query tree has $r = 2^{q(n)}$ paths indexed from 0 through $r - 1$ (see Section 2). On input x, our NP machine N' will guess an index t of a path in the oracle query tree. Next, N' computes the values i and w where $i = r - 1 - t$ and $w = \lfloor a_i \rfloor - p(n)$. Then, N' writes down the m-bit binary representation of w on an auxiliary tape which is used only for this purpose. After this step, N' simulates the computation of $D(x)$ on the path t of the oracle query tree. For each positive query on this path, N' also guesses a satisfying assignment for the query. (If the query is in fact unsatisfiable, N' will reject on all paths that guess t.) Finally, N' determines the value y_t output by $D(x)$ on path t and simulates $N(x, y_t)$ step by step. Recall that N is the NP machine for the second phase of the $\mathrm{NPF}//\mathrm{PF}^{\mathrm{SAT}[q(n)]}$ computation for $f(x)$. On computation paths where $N(x, y_t)$ produces an output, $N'(x)$ will accept. Since f is a total function, $N'(x)$ always accepts.

Now, we use Karp's reduction to convert the computation of $N'(x)$ into a 3CNF formula F. For each tape cell and each time step, this reduction uses a separate variable to encode the fact that the tape cell contains the symbol 1 at that time step. Let the variables $w_0, \ldots, w_{m-1}$ be the variables that determine whether the first m tape cells of the auxiliary tape contains the symbol 1 at the last time step. These were the tape cells where N' wrote down the m-bit binary expansion of the value w. For notational convenience, we let w_0 be the variable representing the cell where the least significant bit of w was written and let w_{m-1} correspond to the most significant bit. In this arrangement, given any satisfying assignment of F, the truth values assigned to $w_0, \ldots, w_{m-1}$ is the binary expansion of the value w.

Next, we take the 3CNF formula F and convert it into an undirected graph G using the standard reduction from SAT to the Hamiltonian Cycle Problem. In this construction, each variable v in F is represented by a pair of paths which share a common start vertex and end vertex. If a Hamiltonian cycle for this graph follows the first path, then that Hamiltonian cycle represents a satisfying assignment for F where the variable v is assigned FALSE. Taking the second path represents an assignment of TRUE to v. The construction of the graph guarantees that any Hamiltonian cycle for the graph must traverse one of the two paths completely and that the vertices on the other path are "picked up" later by the clause gadgets. Call the first edge in the first path the FALSE edge for v and the first edge of the second path the TRUE edge for v. In any Hamiltonian cycle for G exactly one of these two edges are used. Finally, recall the variables $w_0, \ldots, w_{m-1}$ of F that we have identified above. For each w_j, we assign a weight of $2^j + 1$ to the TRUE edge for w_j. All other edges have weight 1. Note that the sum of the weights of all the edges is bounded by

$p(n)^2+2^{m+1}$. Without loss of generality we can assume that $2m < p(n)$, then $p(n)^2+2^{m+1} < 2^{p(n)}$. This completes our construction of the graph G.

We claim that given any $k(n)$-approximate TSP tour of G, we can produce an output of $f(x)$ in polynomial time. First, note that any TSP tour of G corresponds to some satisfying assignment of F, which corresponds to an accepting path of $N'(x)$. On this accepting path, N' computed the value $w = \lfloor a_i \rfloor - p(n)$ for some i with $0 \leq i \leq r-1$. Each of the $p(n)$ edges in the TSP tour contributes a weight of 1, except the TRUE edges which make an *additional* contribution of w. Thus, the total weight of the tour must equal $\lfloor a_i \rfloor$. Since the values of $\lfloor a_i \rfloor$ are separated by a factor of $k(n)$, a $k(n)$-approximate tour of G must in fact be an optimum tour. Recall that i was defined as $r-1-t$ where t is the index of a path in the oracle query tree for $D(x)$ that N' had guessed. Let z be the index of the correct path in the oracle query tree and let $s = r-1-z$. We claim that the weight of the optimum tour in G is $\lfloor a_z \rfloor$.

First, some computation path of $N'(x)$ must accept after guessing z to be the index of the path in the oracle query tree. After guessing z, N' will be able to find satisfying assignments for all the positive queries on that path, since all positive queries on the correct path are satisfiable. Furthermore, since f is a total function, N' will find at least one path of $N(x, y_z)$ which outputs a value. Since z is the index of the correct path, this value is an output of $f(x)$. Moreover, the TSP tour in G which represents this accepting computation of $N'(x)$ will have total length $\lfloor a_z \rfloor$.

Now, suppose that there is a tour with weight $\lfloor a_i \rfloor$ that is less than $\lfloor a_z \rfloor$. Let $t = r-1-i$. Then, the computation path of $N'(x)$ that corresponds to this TSP tour must have guessed path t in the oracle query tree for $D(x)$. Since $\lfloor a_i \rfloor < \lfloor a_z \rfloor$, t must be greater than z. However, since z is the correct path, some positive query on path t must be unsatisfiable. Thus, N' will fail to find a satisfying assignment for this query and will reject. Therefore, there cannot be a TSP tour of G with weight less than $\lfloor a_z \rfloor$.

To complete the proof, we simply note that the edges of an optimum tour in G encodes a satisfying assignment of F and that a satisfying assignment of F encodes an accepting computation path of N'. Thus, given an optimum tour of G, we can, in polynomial time, recover the correct path in the oracle query tree for $D(x)$ *and* find a computation path of $N(x, y_z)$ which outputs a value. Thus, in polynomial time, we can produce an output of $f(x)$ if we are given an optimum tour of G. Finally, observe that for each string α output by $f(x)$, there exists a tour of G given which this procedure will output α. □

Corollary 25

- TSP is $\leq_{\mathbf{m}}^{\mathrm{P}}$-complete for $\mathrm{NPF}^{\mathrm{SAT}[n^{O(1)}]}$.
- $(1+n^{-\log^a n})$-approximating TSP is $\leq_{\mathbf{m}}^{\mathrm{P}}$-complete for $\mathrm{NPF}^{\mathrm{SAT}[O(\log^{a+1} n)]}$.
- $(1+n^{-\log n})$-approximating TSP is $\leq_{\mathbf{m}}^{\mathrm{P}}$-complete for $\mathrm{NPF}^{\mathrm{SAT}[O(\log^2 n)]}$.
- For constant k, k-approximating TSP is $\leq_{\mathbf{m}}^{\mathrm{P}}$-complete for $\mathrm{NPF}^{\mathrm{SAT}[O(\log n)]}$.

Proof: By Lemma 17, $k(n)$-approximating TSP can be accomplished in $\mathrm{NPF}^{\mathrm{SAT}[q(n)]}$ where $q(n) = \log n - \log\log k(n)$. For constant approximation factors k, this procedure uses $\log n - \log\log k$ queries to SAT. For small approximation factors $k(n) = 1+\delta(n)$ that converges rapidly to 1, we can use the Taylor series approximation $\ln(1+\delta(n)) \approx \delta(n)$. Thus, $(1+n^{-\log n})$-approximating TSP uses $\approx \log n - \log n^{-\log n} + \log\ln 2 \approx \log n + \log^2 n$ queries to SAT. By a similar calculation, $(1+n^{-\log^a n})$-approximating TSP uses $O(\log^{a+1} n)$ queries to SAT. Computing the optimum TSP tour uses n queries to SAT since we can determine the length of the shortest TSP tour using binary search.

Now we show that any function f in $\mathrm{NPF}^{\mathrm{SAT}[c\log n]}$ $\leq^{\mathrm{P}}_{\mathbf{m}}$-reduces to any function h that k-approximates TSP for constant k. Using Theorem 24, we simply have to note that in this case, $2^{q(n)}\log k(n) = n^c \log k$ is polynomially bounded. Thus, f reduces to h.

To show that every function f in $\mathrm{NPF}^{\mathrm{SAT}[c\log^2 n]}$ $\leq^{\mathrm{P}}_{\mathbf{m}}$-reduces to any function h that $(1+n^{-\log n})$-approximates TSP, we first use the padding argument in Section 2 to reduce f to a function f' in $\mathrm{NPF}^{\mathrm{SAT}[\log^2 n]}$. Then, by Theorem 24, $f \leq^{\mathrm{P}}_{\mathbf{m}} f' \leq^{\mathrm{P}}_{\mathbf{m}} h$ because

$$2^{\log^2 n}\log(1+n^{-\log n}) \approx n^{\log n}\cdot n^{-\log n} = 1$$

is polynomially bounded. The calculations for functions in $\mathrm{NPF}^{\mathrm{SAT}[c\log^a n]}$ reducing to functions that $(1+n^{-\log^{(a+1)} n})$-approximating TSP are similar. For functions in $\mathrm{NPF}^{\mathrm{SAT}[n^{O(1)}]}$ reducing to functions which compute the optimum TSP tour, we use Theorem 24 with $k(n) = 1+2^{-n}$. Since we bound the total weight of the edges by 2^n, an approximate solution within a factor of $1+2^{-n}$ is in fact an optimum solution. □

5 Hunting for hard sequences

In Lemma 15, we showed that the Boolean Hierarchy collapses if the $\mathrm{NPF}^{\mathrm{SAT}[q(n)]}$ hierarchy collapses. In this section, we show that a partial converse of Lemma 15 also holds. That is, if the Boolean Hierarchy collapses, then so does the $\mathrm{NPF}^{\mathrm{SAT}[q(n)]}$ hierarchy. This is surprising because the Boolean Hierarchy may collapse without collapsing the deterministic bounded query function hierarchy. For example, if NP = coNP but $\mathrm{P} \neq \mathrm{NP}\cap\mathrm{coNP}$, then the Boolean Hierarchy collapses to level 1, but for all constants k, $\mathrm{PF}^{\mathrm{SAT}[k]} \subsetneq \mathrm{PF}^{\mathrm{SAT}[k+1]}$. The main results in this section are:

Theorem 26 Let $q(n) \geq r(n)$ be polynomial-time computable functions such that $q(n)^{r(n)}$ is polynomially bounded. Then,

$$\mathrm{BH}_{r(n)} = \mathrm{coBH}_{r(n)} \Longrightarrow \mathrm{NPF}//\mathrm{PF}^{\mathrm{SAT}[q(n)]}_{\mathrm{tt}} \subseteq \mathrm{NPF}//\mathrm{PF}^{\mathrm{SAT}[r(n)-1]}_{\mathrm{tt}}.$$

Theorem 27 Let $q(n)$ and $r(n)$ be polynomial-time computable functions such that $q(n)$ is polynomially bounded and $r(n) \leq n^{\epsilon}$ for some $\epsilon < 1$. Then,

$$\mathrm{BH}_{r(n)} = \mathrm{coBH}_{r(n)} \Longrightarrow \mathrm{NPF}^{\mathrm{SAT}[q(n)]} \subseteq \mathrm{NPF}//\mathrm{PF}^{\mathrm{SAT}[(r(n)-1)(\log q(n)+2)]}.$$

The proofs of Theorem 26 and 27 are generalizations of recent work by Buhrman and Fortnow [BF96] who showed that $\mathrm{BH}_2 = \mathrm{coBH}_2 \Longrightarrow \mathrm{P}^{\mathrm{SAT}[n^{O(1)}]} \subseteq \mathrm{BH}_2$. These proofs are all modifications of the hard/easy argument which Kadin [Kad88] used to show that PH collapses if BH collapses. We attempt here an explanation of the new advances in this proof technique which allow us to prove our results. Nevertheless, we assume some familiarity with hard/easy arguments in the proofs of these theorems.

Suppose that the Boolean Hierarchy collapses to the second level (i.e., all the way down to D^{P}). The key to the hard/easy argument is the *hard string* — an unsatisfiable formula with special properties. The hard/easy argument places each length n into two cases: either all the unsatisfiable formulas of length n are easy (i.e., not hard) or there exists a hard string. In either case, we have an NP algorithm for $\overline{\mathrm{SAT}}$. This collapses PH, but not to the NP level because we really have $\overline{\mathrm{SAT}} \in \mathrm{NP/poly}$ — for each length n, we need the advice function to provide a hard string or to guarantee that all strings are easy. Subsequent improvements to Kadin's original proof showed tighter collapses of PH [BCO93, CK96]. These proofs are based upon detailed analyses

of the complexity of finding a hard string. The newest improvements to the hard/easy argument circumvents this search entirely. Recently, Hemaspaandra, Hemaspaandra and Hempel [HHH96] showed that

$$\mathrm{P}^{\Sigma_3^{\mathrm{P}}[1]} = \mathrm{P}^{\Sigma_3^{\mathrm{P}}[2]} \Longrightarrow \mathrm{PH} \subseteq \Sigma_3^{\mathrm{P}}.$$

This rather impressive result is actually a *downward* collapse. The proof of this result avoids having to search for a hard string by checking if the input string is a hard string.

By extending this technique, Buhrman and Fortnow were able to show that $\mathrm{D}^{\mathrm{P}} = \text{co-}\mathrm{D}^{\mathrm{P}}$ implies that $\mathrm{P}^{\mathrm{SAT}[n^{O(1)}]} \subseteq \mathrm{D}^{\mathrm{P}}$. Their proof was divided into two parts:

$$\mathrm{D}^{\mathrm{P}} = \text{co-}\mathrm{D}^{\mathrm{P}} \Longrightarrow \mathrm{P}^{\mathrm{SAT}[n^{O(1)}]} \subseteq \mathrm{P}^{\mathrm{SAT}[O(\log n)]} \text{ and } \mathrm{D}^{\mathrm{P}} = \text{co-}\mathrm{D}^{\mathrm{P}} \Longrightarrow \mathrm{P}^{\mathrm{SAT}[O(\log n)]} \subseteq \mathrm{D}^{\mathrm{P}}.$$

The "trick" in these proofs is to consider whether any of the queries used in the $\mathrm{P}^{\mathrm{SAT}[n^{O(1)}]}$ and the $\mathrm{P}^{\mathrm{SAT}[O(\log n)]}$ computation is a hard string (rather than considering whether the input string is a hard string). This again circumvents the need to search for a hard string, because if all the queries are easy strings, an NP computation can simulate the $\mathrm{P}^{\mathrm{SAT}[n^{O(1)}]}$ or the $\mathrm{P}^{\mathrm{SAT}[O(\log n)]}$ computation. On the other hand, if a hard string is found, then we also have a reduction from SAT to $\overline{\mathrm{SAT}}$.

Our proofs for Theorems 26 and 27 are similar. For example, the two theorems put together gives us:

$$\mathrm{D}^{\mathrm{P}} = \text{co-}\mathrm{D}^{\mathrm{P}} \Longrightarrow \mathrm{NPF}^{\mathrm{SAT}[n^{O(1)}]} \subseteq \mathrm{NPF}^{\mathrm{SAT}[1]}$$

The main difficulty here is making sure that the proofs work for non-constant levels of the Boolean Hierarchy and for a non-constant number of queries.

We now provide the proofs of Theorems 26 and 27. Throughout these proofs we assume that $\mathrm{BL}_{r(n)} \leq_{\mathrm{m}}^{\mathrm{P}} \mathrm{coBL}_{r(n)}$ via a polynomial-time reduction h, where $\mathrm{BL}_{r(n)}$ and $\mathrm{coBL}_{r(n)}$ are respectively the complete languages for $\mathrm{BH}_{r(n)}$ and $\mathrm{coBH}_{r(n)}$. The function $r(n)$ will be bounded by n^{ϵ} for some $\epsilon < 1$. The key to the hard/easy arguments that follow is the definition of a hard sequence. We briefly describe the properties of maximal hard sequences that are necessary for our proofs. Detailed justifications of these properties may be found in the literature [Kad88, BCO93, CK96, Wag88, Wag90]. We mainly follow the terminology and notation of Chang and Kadin [CK96].

Definition 28 For $\vec{y} = \langle y_1, \ldots, y_m \rangle$, let $\vec{y}^{\,R} = \langle y_m, \ldots, y_1 \rangle$ be the reversal of the sequence. Let π_k and $\pi_{j,k}$ be the projection functions such that $\pi_k(\vec{y}) = y_k$ and $\pi_{j,k}(\vec{y}) = \langle y_j, \ldots, y_k \rangle$. Furthermore, let $\{0,1\}^{\leq m}$ denote the set of strings over $\{0,1\}$ of length $\leq m$.

Definition 29 Suppose that $\mathrm{BL}_{r(n)} \leq_{\mathrm{m}}^{\mathrm{P}} \mathrm{coBL}_{r(n)}$ via a polynomial-time reduction h. Let $\ell = r(n) - k$. Then, $\vec{x} = \langle x_1, \ldots, x_k \rangle$ is a hard sequence for length m with respect to h, if the following hold:

1. for each i, $1 \leq i \leq k$, $x_i \in \{0,1\}^{\leq m}$.
2. for each i, $1 \leq i \leq k$, $x_i \in \overline{\mathrm{SAT}}$.
3. for all $u_1, \ldots, u_\ell \in \{0,1\}^{\leq m}$, let $\vec{u} = \langle u_1, \ldots, u_\ell \rangle$ and $\langle v_1, \ldots, v_k \rangle = \pi_{\ell+1,r(n)}(h(\vec{u}, \vec{x}^{\,R}))$. Then, $v_i \in \overline{\mathrm{SAT}}$, for all $1 \leq i \leq k$.

We refer to k as the *order* of the hard sequence $\vec{x}$ and for notational convenience, we define the empty sequence to be a hard sequence of order 0.

Given a sequence $\vec{x}$, it is a coNP question to determine whether $\vec{x}$ is a hard sequence. If $\vec{x} = \langle x_1, \ldots, x_k \rangle$ is a hard sequence, then $\pi_{1,\ell}(h(\vec{u}, \vec{x}^{\,R}))$ is a reduction from BL_ℓ to coBL_ℓ on tuples $\langle y_1, \ldots, y_\ell \rangle$ such that the length of each y_i is $\leq m$. We say that a hard sequence $\vec{x}$ is a maximal hard sequence, if for all $w \in \{0,1\}^{\leq m}$, $\langle x_1, \ldots, x_k, w \rangle$ is not a hard sequence. Suppose that $\vec{x}$ is a maximal hard sequence. Then for all $w \in \{0,1\}^{\leq m}$,

$$w \in \overline{\mathrm{SAT}} \Longrightarrow \exists u_1, \ldots, u_{\ell-1} \in \{0,1\}^{\leq m}, \pi_\ell(h(u_1, \ldots, u_{\ell-1}, w, \vec{x}^{\,R})) \in \mathrm{SAT}.$$

If this were not the case, then $\langle x_1, \ldots, x_k, w \rangle$ would be a hard sequence, violating the maximality of $\vec{x}$. Furthermore, for all $w \in \{0,1\}^{\leq m}$,

$$w \in \mathrm{SAT} \Longrightarrow \forall u_1, \ldots, u_{\ell-1} \in \{0,1\}^{\leq m}, \pi_\ell(h(u_1, \ldots, u_{\ell-1}, w, \vec{x}^{\,R})) \in \overline{\mathrm{SAT}}.$$

This follows from the definitions of BL_ℓ and coBL_ℓ. Thus, a maximal hard sequence for length m provides an NP algorithm for $\overline{\mathrm{SAT}}$ on $\{0,1\}^{\leq m}$. Moreover, given a hard sequence $\vec{x}$, we say that a string w is *easy* with respect to $\vec{x}$ if $|w| \leq m$ and there exists $u_1, \ldots, u_{\ell-1} \in \{0,1\}^{\leq m}$ such that

$$\pi_\ell(h(u_1, \ldots, u_{\ell-1}, w, \vec{x}^{\,R})) \in \mathrm{SAT}.$$

This condition, along with the definitions of BL_ℓ and coBL_ℓ, guarantees that w must be in $\overline{\mathrm{SAT}}$. If $\vec{x}$ is known to be a hard sequence, then for each w that is easy with respect to $\vec{x}$, there is an existential witness for $w \in \overline{\mathrm{SAT}}$ — namely $\vec{u} = \langle u_1, \ldots, u_{\ell-1} \rangle$ and a satisfying assignment for $\pi_\ell(h(\vec{u}, w, \vec{x}^{\,R}))$.

Since an NP machine can recognize $\overline{\mathrm{SAT}}$ using only one maximal hard sequence for each length n, a polynomial length advice function can provide this information. Thus, $\mathrm{BL}_{r(n)} \leq^{\mathrm{P}}_{\mathrm{m}} \mathrm{coBL}_{r(n)}$ implies that coNP $\subseteq$ NP/poly. This collapses PH to Σ^{P}_3 using Yap's theorem [Yap83]. Now we show that the $\mathrm{NPF}^{\mathrm{SAT}[q(n)]}$ hierarchy also collapses.

Proof of Theorem 26:

Let f be a function in $\mathrm{NPF}//\mathrm{PF}^{\mathrm{SAT}[q(n)]}_{\mathrm{tt}}$ via a deterministic polynomial time machine D and an NP machine N. We construct an equivalent pair of machines N' and D' which ask $r(n) - 1$ queries to SAT. Fix an input string x of length n. Let $W = \{ w_1, \ldots, w_{q(n)} \}$ be the set of parallel queries asked by $D(x)$ and let m be the length of the longest query. Let $\mathrm{HARD}(m, W)$ be the set of hard sequences for length m where every component of each hard sequence is a string from W. For each k, $1 \leq k < r(n)$, $D'(x)$ asks its SAT oracle whether there exists a hard sequence of order k in $\mathrm{HARD}(m, W)$. Since $q(n)^{r(n)}$ is polynomially bounded, there are only polynomially many sequences in $\mathrm{HARD}(m, W)$. Recall that it is a coNP question to determine whether a particular sequence is a hard sequence. Thus, asking whether one of polynomially many candidates is a hard sequence is also a coNP question. Therefore, using at most $r(n) - 1$ queries, $D'(x)$ can compute the maximum order of the hard sequences in $\mathrm{HARD}(m, W)$. This value, call it z, is the output of $D'(x)$.

On input (x, z), the machine N' will proceed as follows. First, N' guesses two sets W_{SAT} and H. The set W_{SAT} is a subset W. If N' guesses W_{SAT} correctly, then W_{SAT} would be exactly $W \cap \mathrm{SAT}$. The set H is a set of sequences of strings from W. If N' guesses H correctly, then H would be exactly $\mathrm{HARD}(m, W)$. Thus, the sequences in H should not have more than z components.

As usual, N' might guess W_{SAT} and H incorrectly. So, N' performs the following verifications on W_{SAT} and H. For each $w \in W_{\mathrm{SAT}}$, N' confirms that w is satisfiable by guessing a satisfying assignment for w. It remains possible that some $w \in W - W_{\mathrm{SAT}}$ is satisfiable. Next, we try to verify that each sequence $\vec{y} = \langle y_1, \ldots, y_k \rangle \in H$ is a hard sequence. First, each y_i must be an element of $W - W_{\mathrm{SAT}}$, since the components of a hard sequence must be unsatisfiable. Also, for each $\vec{y} = \langle y_1, \ldots, y_k \rangle \in H$ and each $w \in W - W_{\mathrm{SAT}}$, if $\langle \vec{y}, w \rangle \notin H$, then w should be easy with respect to $\vec{y}$. This can be confirmed using the following NP procedure:

PROCEDURE EasyWitness($\langle y_1, \ldots, y_k \rangle, w$)

1. Let $\ell = r(n) - k$
2. Guess a sequence $u_1, \ldots, u_{\ell-1} \in \{0,1\}^{\leq m}$
3. Compute the formula $F = \pi_\ell(h(u_1, \ldots, u_{\ell-1}, w, \vec{y}^R)$
4. Guess a satisfying assignment for F.

Clearly, if $W_{\rm SAT} = W \cap {\rm SAT}$ and $H = {\rm HARD}(m, W)$, then each verification step will succeed. We claim that if $W_{\rm SAT}$ and H pass every verification step, then $W_{\rm SAT} = W \cap {\rm SAT}$.

Suppose that H passes every verification step. Let $\vec{x} = \langle x_1, \ldots, x_z \rangle$ be a hard sequence where each $x_i \in W - W_{\rm SAT}$. We claim that $\vec{x}$ must be in H. Suppose not. Without loss of generality we can assume that the empty sequence is in H, since the empty sequence is by definition a hard sequence. Then there exist i, $0 \leq i < z$ such that $\langle x_1, \ldots, x_i \rangle \in H$ but $\langle x_1, \ldots, x_{i+1} \rangle \notin H$. Then, x_{i+1} should be easy with respect to the hard sequence $\langle x_1, \ldots, x_i \rangle$. This will prompt N' to run the EasyWitness procedure on $\langle x_1, \ldots, x_i \rangle$ and x_{i+1}. However, $\langle x_1, \ldots, x_{i+1} \rangle$ is in reality a hard sequence, so EasyWitness($\langle x_1, \ldots, x_i \rangle, x_{i+1}$) will fail. Thus, H would not have passed every verification, which is a contradiction. Therefore, H must contain $\vec{x}$.

Now, suppose that $W_{\rm SAT}$ and H passes every verification step. Fix a string $w \in W - W_{\rm SAT}$. By the preceding argument H contains at least one hard sequence of order z. We know that such a hard sequence exists since z was computed by $D'(x)$ to be the maximum order of the hard sequences in ${\rm HARD}(m, W)$. Let $\vec{x}$ be such a maximum hard sequence. Since $\langle \vec{x}, w \rangle \notin H$, N' must have succeeded in the procedure call EasyWitness($\vec{x}, w$). Then, there exists $u_1, \ldots, u_{\ell-1} \in \{0,1\}^{\leq m}$ such that $\pi_\ell(h(u_1, \ldots, u_{\ell-1}, w, \vec{x}^R) \in {\rm SAT}$, where $\ell = r(n) - z$. By the definitions of ${\rm BL}_{r(n)}$ and ${\rm coBL}_{r(n)}$, this is enough to imply that $w \in \overline{\rm SAT}$. Thus, every string $w \in W - W_{\rm SAT}$ must be unsatisfiable. Since every string in $W_{\rm SAT}$ was already confirmed to be satisfiable, it follows that $W_{\rm SAT} = W \cap {\rm SAT}$.

Finally, some computation path of N' will guess the correct $W_{\rm SAT}$ and the correct H which passes every verification step. On such a path, N' knows the elements of $W \cap {\rm SAT}$. Thus, N' can carry out the simulation of N and D step by step without using any queries to SAT. □

Proof of Theorem 27:

Let f be a function in ${\rm NPF}^{{\rm SAT}[q(n)]}$. By Lemma 10, $f \in {\rm NPF}//{\rm PF}^{{\rm SAT}[2q(n)]}$ via a deterministic polynomial time machine D and an NP machine N. Fix an input string x of length n. Then, $D(x)$ asks at most $2q(n)$ serial queries to the SAT oracle. This computation can be represented by an oracle query tree of depth $2q(n)$. Let m be a bound on the length of the queries. Since $q(n)$ may be linear in n, there could be exponentially many nodes in the oracle query tree. Now, suppose that every query in the tree is either satisfiable or easy with respect to the empty sequence. Then, an NP machine N_0 can determine the correct path in the oracle query tree for $D(x)$. When $D(x)$ asks a query w, N_0 will simply guess whether w is satisfiable or unsatisfiable but easy. In either case, N_0 can verify whether its guess is correct. If w turns out to be hard, then all computation paths of N_0 will reject. This is not a bad situation, since locating a hard string is generally useful.

Now, assuming nothing about the queries in the oracle query tree, we will build a machine N_1 to help us look for the location of the first query on the correct path that is hard with respect to the empty sequence (i.e., an unsatisfiable formula that is not easy with respect to the empty sequence). The machine N_1 is given the original input x and a level number i and accepts if every query on the correct path up to level i is either satisfiable or easy. Then, using binary search and $\log(2q(n))$ queries to SAT, a polynomial-time machine D' can determine the level where the first

hard string appeared on the correct path. Note that D' does not have the hard string itself or its exact location in the oracle query tree. It only has the level where the first hard string occurred.

We construct an NP machine N' and a $\mathrm{PF}^{\mathrm{SAT}[(r(n)-1)(\log q(n)+2)]}$ machine D' to show that $f \in \mathrm{NPF}//\mathrm{PF}^{\mathrm{SAT}[(r(n)-1)(\log q(n)+2)]}$. The machine D' will expend its queries in $r(n)-1$ rounds. In each round, it uses $\lceil \log(2q(n)) \rceil \leq \log q(n) + 2$ queries to SAT to locate a level where a hard string occurs. In the first round, $D'(x)$ uses N_1 to determine the first level where a hard string occurs on the correct path in the oracle query tree for $D(x)$, as described above. Call this level i_1. In the second round, $D'(x)$ will use an NP machine N_2 to locate the level where the second component of a hard sequence occurs. The NP machine N_2 will be given i_1 as part of its input. Then N_2 knows that every query on the correct path up to level i_1 is either satisfiable or easy. Thus, N_2 can answer the queries up to level $i_1 - 1$. Then, N_2 simulates $D(x)$ until $D(x)$ asks the i_1th query x_1. Thus, $\langle x_1 \rangle$ is a hard sequence of order 1. By definition of a hard sequence, $x_1 \in \overline{\mathrm{SAT}}$. So, N_2 knows the correct path in the oracle query tree up to level i_1. Then, N_2 can proceed to determine whether every query between level i_1 and some given level i is either satisfiable or easy with respect to $\langle x_1 \rangle$. Thus, using N_2 and another round of $\log q(n) + 2$ queries, $D'(x)$ can determine the level i_2 where the second component of a hard sequence occurs. After some number of rounds $k \leq r(n) - 1$, all the remaining queries on the correct path will be easy with respect to $\langle x_1, \ldots, x_k \rangle$ and $D'(x)$ will have determined $i_1, \ldots, i_k$, the levels where the components of a maximal hard sequence occurred. Finally, $D'(x)$ gives the values of $i_1, \ldots, i_k$ to N' and N' can complete the simulation of $D(x)$ to compute its output. After that, N' will simply simulate the original machine N step by step to compute f. □

The combination of Theorems 26 and 27 provides us with some results about the Boolean Hierarchy. First, we show that if the Boolean Hierarchy collapses to a constant level, then so does the nondeterministic bounded query hierarchy.

Theorem 30 If $\mathrm{BH}_{2^k} = \mathrm{coBH}_{2^k}$ then $\mathrm{NPF}^{\mathrm{SAT}[n^{O(1)}]} \subseteq \mathrm{NPF}^{\mathrm{SAT}[k]}$.

Proof: By assumption, $\mathrm{BL}_{2^k} \leq^{\mathrm{P}}_{\mathrm{m}} \mathrm{coBL}_{2^k}$. Fix a constant d and let f be a multi-valued function in $\mathrm{NPF}^{\mathrm{SAT}[n^d]}$. We apply Theorem 26 with $r(n) = 2^k$ and $q(n) = n^c$ where $c = 2^k d + 1$ to obtain

$$\mathrm{NPF}//\mathrm{PF}^{\mathrm{SAT}[n^c]}_{\mathrm{tt}} \subseteq \mathrm{NPF}//\mathrm{PF}^{\mathrm{SAT}[2^k-1]}_{\mathrm{tt}}.$$

The results from Theorem 26 are for parallel queries to SAT. However, by Lemma 13,

$$\mathrm{NPF}//\mathrm{PF}^{\mathrm{SAT}[c \log n]} \subseteq \mathrm{NPF}//\mathrm{PF}^{\mathrm{SAT}[n^c]}_{\mathrm{tt}} \quad \text{and} \quad \mathrm{NPF}//\mathrm{PF}^{\mathrm{SAT}[2^k-1]}_{\mathrm{tt}} = \mathrm{NPF}^{\mathrm{SAT}[k]}.$$

Next, we apply Theorem 27 with $q(n) = n^d$ and $r(n) = k$ to get

$$\mathrm{NPF}^{\mathrm{SAT}[n^d]} \subseteq \mathrm{NPF}//\mathrm{PF}^{\mathrm{SAT}[(2^k-1)(\log n^d+2)]}.$$

Since $(2^k - 1)(\log n^d + 2) < (2^k d + 1) \log n$ for large enough n, we have the desired collapse

$$\mathrm{NPF}^{\mathrm{SAT}[n^d]} \subseteq \mathrm{NPF}//\mathrm{PF}^{\mathrm{SAT}[c \log n]} \subseteq \mathrm{NPF}//\mathrm{PF}^{\mathrm{SAT}[n^c]}_{\mathrm{tt}} \subseteq \mathrm{NPF}^{\mathrm{SAT}[k]}.$$

□

The collapse of $\mathrm{NPF}^{\mathrm{SAT}[n^{O(1)}]}$ all the way down to $\mathrm{NPF}^{\mathrm{SAT}[k]}$ is quite drastic. For example, by the results in Section 4, such a collapse implies that solving TSP exactly can be reduced to 2-approximating MaxClique. Conversely, if TSP does not reduce to 2-approximating MaxClique, then the Boolean Hierarchy must have infinitely many distinct levels. Thus, we have the following corollary.

Corollary 31 If any of the following conditions holds, then the Boolean Hierarchy is infinite — i.e., for all constants k, $\mathrm{BH}_k \subsetneq \mathrm{BH}_{k+1}$.

- TSP $\not\leq^{\mathrm{P}}_{\mathrm{m}}$ MaxClique
- TSP $\not\leq^{\mathrm{P}}_{\mathrm{m}}$ 2-approximating TSP
- MaxClique $\not\leq^{\mathrm{P}}_{\mathrm{m}}$ 2-approximating MaxClique
- TSP $\not\leq^{\mathrm{P}}_{\mathrm{m}}$ 2-approximating MaxClique

Two types of results are usually cited as "evidence" that the Boolean Hierarchy is infinite. First, if we assume that PH is infinite, then the Boolean Hierarchy is infinite [Kad88]. Alternatively, one might resort to the Random Oracle Hypothesis, since under random oracles the Boolean Hierarchy is infinite with probability 1 [Cai87]. Here we offer another option. Assuming that certain NP-approximation problems really are harder than others (e.g., MaxClique versus 2-approximating MaxClique), then we can show that the Boolean Hierarchy is infinite. This is an arguably more natural justification for an infinite Boolean Hierarchy.

Finally, the results of Section 5 also imply a complete upward collapse of the Boolean Hierarchy and the bounded query hierarchy at the constant levels. These corollaries generalize the result of Buhrman and Fortnow [BF96] which showed that $\mathrm{P}^{\mathrm{SAT}[1]} = \mathrm{P}^{\mathrm{SAT}[2]} \Longrightarrow \mathrm{P}^{\mathrm{SAT}[n^{O(1)}]} \subseteq \mathrm{P}^{\mathrm{SAT}[1]}$.

Corollary 32 For all constants k, $\mathrm{BH}_k = \mathrm{coBH}_k \Longrightarrow \mathrm{P}^{\mathrm{SAT}[n^{O(1)}]} \subseteq \mathrm{BH}_k$.

Proof: Let 2^c be the smallest power of 2 that is $\geq k$. Using the standard upward collapse of the Boolean Hierarchy [CGH+88], we know that for all constants $k' > k$,

$$\mathrm{BH}_k = \mathrm{coBH}_k = \mathrm{BH}_{k'} = \mathrm{coBH}_{k'}.$$

Thus, by Theorem 30, $\mathrm{NPF}^{\mathrm{SAT}[n^{O(1)}]} \subseteq \mathrm{NPF}^{\mathrm{SAT}[c]}$. Since the characteristic function of every language in $\mathrm{P}^{\mathrm{SAT}[n^{O(1)}]}$ is computable in $\mathrm{NPF}^{\mathrm{SAT}[n^{O(1)}]}$, $\mathrm{P}^{\mathrm{SAT}[n^{O(1)}]} \subseteq \mathrm{NP}^{\mathrm{SAT}[c]}$. By Lemmas 13 and 14,

$$\mathrm{NP}^{\mathrm{SAT}[c]} = \mathrm{NP}//\mathrm{PF}^{\mathrm{SAT}[2^c-1]}_{\mathrm{tt}} = \mathrm{BH}_{2^{c+1}+1}.$$

Since $2^{c+1}+1 > k$, by the standard upward collapse of the Boolean Hierarchy, $\mathrm{BH}_k = \mathrm{BH}_{2^{c+1}+1}$. Therefore, $\mathrm{P}^{\mathrm{SAT}[n^{O(1)}]} \subseteq \mathrm{BH}_k$. □

Since the Boolean Hierarchy and the bounded query language classes are intertwined, we also have a complete upward collapse of the bounded query language classes. The corollary follows from this fact [CGH+88, Bei91]:

$$\mathrm{P}^{\mathrm{SAT}[k]} \subseteq \mathrm{BH}_{2^k} \cap \mathrm{coBH}_{2^k} \subseteq \mathrm{BH}_{2^k} \cup \mathrm{coBH}_{2^k} \subseteq \mathrm{P}^{\mathrm{SAT}[k+1]}.$$

Corollary 33 For all constants k, if $\mathrm{P}^{\mathrm{SAT}[k]} = \mathrm{P}^{\mathrm{SAT}[k+1]}$ then $\mathrm{P}^{\mathrm{SAT}[n^{O(1)}]} \subseteq \mathrm{P}^{\mathrm{SAT}[k]}$.

6 Upward collapse

Combining the results in Sections 4 and 5, we can obtain upward collapses for the $\mathrm{NPF}^{\mathrm{SAT}[q(n)]}$ hierarchy and for some NP-approximation problems. The proofs of the lemmas and corollaries demonstrate the connection between reducibility among NP-approximation problems and the upward collapse of the nondeterministic bounded query classes.

Lemma 34 Let $f(n) \leq \log\log n + O(1)$ be a polynomial-time computable function. Then,

$$\mathrm{NPF}^{\mathrm{SAT}[f(n)]} = \mathrm{NPF}^{\mathrm{SAT}[f(n)+1]} \Longrightarrow \mathrm{NPF}^{\mathrm{SAT}[n^{O(1)}]} \subseteq \mathrm{NPF}^{\mathrm{SAT}[O(\log^2 n)]}.$$

Proof: By Lemma 15, the Boolean Hierarchy collapses to a level $r(n) \in O(\log n)$. Applying Theorem 27 with $q(n) = n^{O(1)}$, we get $\mathrm{NPF}^{\mathrm{SAT}[n^{O(1)}]} \subseteq \mathrm{NPF}//\mathrm{PF}^{\mathrm{SAT}[O(\log^2 n)]}$. By Lemma 10, $\mathrm{NPF}//\mathrm{PF}^{\mathrm{SAT}[O(\log^2 n)]} = \mathrm{NPF}^{\mathrm{SAT}[O(\log^2 n)]}$. □

Corollary 35 If MaxClique $\leq^{\mathrm{P}}_{\mathrm{m}}$ 2-approximating MaxClique, then

$$\mathrm{TSP} \equiv^{\mathrm{P}}_{\mathrm{m}} (1 + n^{-\log n})\text{-approximating TSP}.$$

Proof: By Corollary 21, we know that MaxClique is complete for $\mathrm{NPF}^{\mathrm{SAT}[O(\log n)]}$ and we know that 2-approximating MaxClique is complete for $\mathrm{NPF}^{\mathrm{SAT}[\log\log n + O(1)]}$. Thus, for some query bound $f(n) \leq \log\log n + O(1)$, we have $\mathrm{NPF}^{\mathrm{SAT}[f(n)]} = \mathrm{NPF}^{\mathrm{SAT}[f(n)+1]}$. Then, by Lemma 34, $\mathrm{NPF}^{\mathrm{SAT}[n^{O(1)}]} \subseteq \mathrm{NPF}^{\mathrm{SAT}[O(\log^2 n)]}$. Finally, by Corollary 25, $\mathrm{TSP} \equiv^{\mathrm{P}}_{\mathrm{m}} (1 + n^{-\log n})$-approximating TSP. □

In the proof of Lemma 34, we could repeatedly apply Theorem 27 and get a lower collapse of $\mathrm{NPF}^{\mathrm{SAT}[n^{O(1)}]}$ (say, down to $\log n \log\log\log n$ queries instead of $\log^2 n$ queries). However, the theorem can only be used a constant number of times. Thus, when $f(n) = \log\log n + O(1)$, we cannot get a collapse of $n^{O(1)}$ queries down to $O(\log n)$ queries. We need a slightly smaller $f(n)$ for this.

Lemma 36 Let $f(n) \leq \log\log n - \log\log\log n + O(1)$ be a polynomial-time computable function. If $\mathrm{NPF}^{\mathrm{SAT}[f(n)]} = \mathrm{NPF}^{\mathrm{SAT}[f(n)+1]}$, then

$$\mathrm{NPF}^{\mathrm{SAT}[n^{O(1)}]} \subseteq \mathrm{NPF}^{\mathrm{SAT}[O(\log n)]} \quad \text{and} \quad \mathrm{NPF}^{\mathrm{SAT}[O(\log\log n)]} \subseteq \mathrm{NPF}^{\mathrm{SAT}[f(n)]}.$$

Proof: By Lemma 15, the Boolean Hierarchy collapses to level $r(n) = 2^{f(n)+1} - 1$. For some constant k, $f(n) \leq \log\log n - \log\log\log n + \log k$, so $r(n) \leq k'(\log n)/(\log\log n)$ for some constant k'.

First, we prove that $\mathrm{NPF}^{\mathrm{SAT}[O(\log\log n)]} \subseteq \mathrm{NPF}^{\mathrm{SAT}[f(n)]}$. Let $q_1(n) = \log^c n$, then by Lemma 13

$$\mathrm{NPF}^{\mathrm{SAT}[c\log\log n]} \subseteq \mathrm{NPF}//\mathrm{PF}^{\mathrm{SAT}[q_1(n)]}_{\mathrm{tt}}.$$

Furthermore, by elementary algebra, $q_1(n)^{r(n)}$ equals $n^{ck'}$ and is polynomially bounded. Thus, we can apply Theorem 26 and get $\mathrm{NPF}//\mathrm{PF}^{\mathrm{SAT}[q_1(n)]}_{\mathrm{tt}} \subseteq \mathrm{NPF}//\mathrm{PF}^{\mathrm{SAT}[r(n)-1]}_{\mathrm{tt}}$. Observe that

$$\mathrm{NPF}//\mathrm{PF}^{\mathrm{SAT}[r(n)-1]}_{\mathrm{tt}} = \mathrm{NPF}//\mathrm{PF}^{\mathrm{SAT}[2^{f(n)+1}-2]}_{\mathrm{tt}} \subseteq \mathrm{NPF}^{\mathrm{SAT}[f(n)+1]}.$$

By assumption $\mathrm{NPF}^{\mathrm{SAT}[f(n)+1]} \subseteq \mathrm{NPF}^{\mathrm{SAT}[f(n)]}$. Thus,

$$\mathrm{NPF}^{\mathrm{SAT}[c\log\log n]} \subseteq \mathrm{NPF}//\mathrm{PF}^{\mathrm{SAT}[\log^c n]}_{\mathrm{tt}} \subseteq \mathrm{NPF}^{\mathrm{SAT}[f(n)]}. \tag{2}$$

Next, we prove that $\mathrm{NPF}^{\mathrm{SAT}[n^{O(1)}]} \subseteq \mathrm{NPF}^{\mathrm{SAT}[O(\log n)]}$. Let $q_2(n) = n^c$. By Theorem 27,

$$\mathrm{NPF}^{\mathrm{SAT}[n^c]} \subseteq \mathrm{NPF}//\mathrm{PF}^{\mathrm{SAT}[r(n)(c\log n+2)]}.$$

Since $r(n) \leq k'(\log n)/(\log\log n)$, $r(n)(\log n^c + 2)$ is bounded by $\log^2 n$. Thus,

$$\mathrm{NPF}^{\mathrm{SAT}[n^c]} \subseteq \mathrm{NPF}//\mathrm{PF}^{\mathrm{SAT}[\log^2 n]}. \tag{3}$$

Then, we apply Theorem 27 with $q_3(n) = \log^2(n)$ and get

$$\mathrm{NPF}^{\mathrm{SAT}[\log^2 n]} \subseteq \mathrm{NPF}//\mathrm{PF}^{\mathrm{SAT}[r(n)(2\log\log n+2)]}.$$

Since $r(n) \leq k'(\log n)/(\log\log n)$, we have

$$\mathrm{NPF}^{\mathrm{SAT}[\log^2 n]} \subseteq \mathrm{NPF}//\mathrm{PF}^{\mathrm{SAT}[3k'\log n]}. \tag{4}$$

Combining Equations 3 and 4, we have $\mathrm{NPF}^{\mathrm{SAT}[n^c]} \subseteq \mathrm{NPF}//\mathrm{PF}^{\mathrm{SAT}[c'\log n]}$. □

Corollary 37 If MaxClique $\leq^{\mathrm{P}}_{\mathrm{m}}$ $(\log n)$-approximating MaxClique, then

$$\text{TSP} \equiv^{\mathrm{P}}_{\mathrm{m}} \text{MaxClique} \equiv^{\mathrm{P}}_{\mathrm{m}} (\log n)\text{-approximating MaxClique}.$$

Proof: By Corollary 21, we know that MaxClique is complete for $\mathrm{NPF}^{\mathrm{SAT}[O(\log n)]}$ and $(\log n)$-approximating MaxClique is complete for $\mathrm{NPF}^{\mathrm{SAT}[\log\log n-\log\log\log n+O(1)]}$. Therefore, under the assumption that MaxClique $\leq^{\mathrm{P}}_{\mathrm{m}}$ $(\log n)$-approximating MaxClique,

$$\mathrm{NPF}^{\mathrm{SAT}[O(\log n)]} = \mathrm{NPF}^{\mathrm{SAT}[\log\log n-\log\log\log n+O(1)]}.$$

By Lemma 36, $\mathrm{NPF}^{\mathrm{SAT}[n^{O(1)}]} = \mathrm{NPF}^{\mathrm{SAT}[O(\log n)]}$. Therefore,

$$\mathrm{NPF}^{\mathrm{SAT}[n^{O(1)}]} = \mathrm{NPF}^{\mathrm{SAT}[\log\log n-\log\log\log n+O(1)]}.$$

Then, by the completeness of TSP, MaxClique and $(\log n)$-approximating MaxClique, the three problems are $\leq^{\mathrm{P}}_{\mathrm{m}}$-equivalent. □

In Lemma 36, we do not have a complete collapse of the $\mathrm{NPF}^{\mathrm{SAT}[q(n)]}$ hierarchy — there is a gap between $O(\log n)$ and $O(\log\log n)$ queries where the classes may be different. To get a complete upward collapse, we need to have an even smaller $f(n)$.

Lemma 38 Let $f(n) \leq \log\log\log n + O(1)$ be a polynomial-time computable function. Suppose that $\mathrm{NPF}^{\mathrm{SAT}[f(n)]} = \mathrm{NPF}^{\mathrm{SAT}[f(n)+1]}$, then $\mathrm{NPF}^{\mathrm{SAT}[n^{O(1)}]} \subseteq \mathrm{NPF}^{\mathrm{SAT}[f(n)]}$.

Proof: Under these assumptions, the Boolean Hierarchy collapses to level $r(n) = 2^{f(n)+1} - 1$ by Lemma 15. For some constant k, $r(n) \leq k\log\log n$. We first apply Theorem 26 with

$$q(n) = (\log n)^{(\log\log n)^{d-1}}.$$

By elementary algebra, $q(n)^{r(n)}$ is polynomially bounded. Thus,

$$\mathrm{NPF}^{\mathrm{SAT}[(\log\log n)^d]} \subseteq \mathrm{NPF}//\mathrm{PF}^{\mathrm{SAT}[q(n)]}_{\mathrm{tt}} \subseteq \mathrm{NPF}//\mathrm{PF}^{\mathrm{SAT}[r(n)-1]}_{\mathrm{tt}} \subseteq \mathrm{NPF}^{\mathrm{SAT}[f(n)]}. \tag{5}$$

Then, we apply Theorem 27 twice with $q(n) = n^c$ and $q(n) = \log^2 n$ and get:

$$\mathrm{NPF}^{\mathrm{SAT}[n^c]} \subseteq \mathrm{NPF}^{\mathrm{SAT}[r(n)(c\log n+2)]} \subseteq \mathrm{NPF}^{\mathrm{SAT}[\log^2 n]} \tag{6}$$

$$\mathrm{NPF}^{\mathrm{SAT}[\log^2 n]} \subseteq \mathrm{NPF}^{\mathrm{SAT}[r(n)(2\log\log n+2)]} \subseteq \mathrm{NPF}^{\mathrm{SAT}[(\log\log n)^3]}. \tag{7}$$

Combining Equations 5, 6 and 7, for all c, $\mathrm{NPF}^{\mathrm{SAT}[n^c]} \subseteq \mathrm{NPF}^{\mathrm{SAT}[f(n)]}$. □

Corollary 39 If 2-approximating MaxClique $\leq^{\mathrm{P}}_{\mathrm{m}}$ 2-approximating Set Cover, then

$$\text{TSP} \equiv^{\mathrm{P}}_{\mathrm{m}} \text{MaxClique} \equiv^{\mathrm{P}}_{\mathrm{m}} \text{2-approximating MaxClique}.$$

Proof: Since 2-approximating Set Cover can be solved in $\mathrm{NPF}^{\mathrm{SAT}[\log\log\log n+O(1)]}$ (we do not have completeness here), by hypothesis, 2-approximating MaxClique can also be solved using $\log\log\log n + O(1)$ queries. Using the completeness of 2-approximating MaxClique and Lemma 38, we have $\mathrm{NPF}^{\mathrm{SAT}[n^{O(1)}]} \subseteq \mathrm{NPF}^{\mathrm{SAT}[\log\log\log n+O(1)]}$. Hence, TSP $\equiv^{\mathrm{P}}_{\mathrm{m}}$ MaxClique $\equiv^{\mathrm{P}}_{\mathrm{m}}$ 2-approximating MaxClique. □

7 Conclusion

We have illustrated an intricate connection linking the complexity of NP-approximation problems, the $\mathrm{NPF}^{\mathrm{SAT}[q(n)]}$ hierarchy and the Boolean Hierarchy. These links are quite strong. For example, for all constants k we have

$$\mathrm{NPF}^{\mathrm{SAT}[k]} = \mathrm{NPF}^{\mathrm{SAT}[k+1]} \Longrightarrow \mathrm{BH}_{2^{k+1}-1} = \mathrm{coBH}_{2^{k+1}-1}$$

$$\mathrm{BH}_{2^k} = \mathrm{coBH}_{2^k} \Longrightarrow \mathrm{NPF}^{\mathrm{SAT}[k]} = \mathrm{NPF}^{\mathrm{SAT}[k+1]}.$$

It is an open question whether $\mathrm{BH}_{2^{k+1}-1} = \mathrm{coBH}_{2^{k+1}-1}$ is sufficient to collapse $\mathrm{NPF}^{\mathrm{SAT}[k]} = \mathrm{NPF}^{\mathrm{SAT}[k+1]}$, but these statements do show that the Boolean Hierarchy collapses if and only if the $\mathrm{NPF}^{\mathrm{SAT}[k]}$ hierarchy collapses.

The link between the NP-approximation problems and the $\mathrm{NPF}^{\mathrm{SAT}[q(n)]}$ hierarchy is also very strong. The results we have proven are completeness results, so any class that captures the complexity of these NP-approximation problems must also capture the properties of the $\mathrm{NPF}^{\mathrm{SAT}[q(n)]}$ machines. For example, any class $\mathcal{C}$ that contains a function which 2-approximates MaxClique, also contains a function that is hard for $\mathrm{NPF}^{\mathrm{SAT}[O(\log\log n)]}$. In Section 4, we have not listed all the known completeness results. We can prove similar theorems for Graph Coloring and MaxSAT. Thus, we consider the upward collapse results presented here as a list of examples rather than an exhaustive list of all known upward collapses.

As for open questions, it would be nice if we could strengthen some of our upward collapse results. For example, we showed in Lemma 34 that

$$\mathrm{NPF}^{\mathrm{SAT}[\log\log n]} \subseteq \mathrm{NPF}^{\mathrm{SAT}[\log\log n+1]} \Longrightarrow \mathrm{NPF}^{\mathrm{SAT}[n^{O(1)}]} \subseteq \mathrm{NPF}^{\mathrm{SAT}[O(\log^2 n)]}.$$

Under the assumption that MaxClique $\leq^{\mathrm{P}}_{\mathrm{m}}$ 2-approximating MaxClique, this leaves a gap between $O(\log^2 n)$ and $O(\log n)$ queries. If we can improve our results and show that

$$\mathrm{NPF}^{\mathrm{SAT}[\log\log n]} \subseteq \mathrm{NPF}^{\mathrm{SAT}[\log\log n+1]} \Longrightarrow \mathrm{NPF}^{\mathrm{SAT}[n^{O(1)}]} \subseteq \mathrm{NPF}^{\mathrm{SAT}[O(\log n)]},$$

then we could claim the rather nice upward collapse:

$$\text{MaxClique} \leq^{\mathrm{P}}_{\mathrm{m}} \text{2-approximating MaxClique} \Longrightarrow \text{TSP} \equiv^{\mathrm{P}}_{\mathrm{m}} \text{2-approximating TSP}.$$

However, this improvement is beyond the reach of current techniques. We conjecture that there is a yet undiscovered variation of the hard/easy argument which can produce the desired results.

Acknowledgments

The author would like to thank Lance Fortnow and Harry Buhrman for providing the beginnings of Theorems 26 and 27. This paper also benefited from discussions with Madhu Sudan and Luca Trevisan.

References

[ABG90] 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.

[AFWZ95] N. Alon, U. Feige, A. Wigderson, and D. Zuckerman. Derandomized graph products. *Computational Complexity*, 5:60–75, 1995.

[ALM+92] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and hardness of approximation problems. In *Proceedings of the IEEE Symposium on Foundations of Computer Science*, pages 14–23, 1992.

[Aro94] S. Arora. *Probabilistic Checking of Proofs and Hardness of Approximation Problems.* PhD thesis, University of California Berkeley, August 1994. Revised version available as Technical Report CS-TR-476-94.

[AS92] S. Arora and S. Safra. Probabilistic checking of proofs. In *Proceedings of the IEEE Symposium on Foundations of Computer Science*, pages 2–13, 1992.

[BC97] R. Beigel and R. Chang. Commutative queries. In *Proccedings of the 5th Israel Symposium on Theory of Computing and Systems*, pages 159–165. IEEE Computer Science Press, June 1997.

[BCO93] 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.

[Bei91] R. Beigel. Bounded queries to SAT and the Boolean hierarchy. *Theoretical Computer Science*, 84(2):199–223, July 1991.

[BF96] H. Buhrman and L. Fortnow. Two queries. Technical Report TR CS 96-20, Department of Computer Science, University of Chicago, 1996.

[BLS84] R. V. Book, T. J. Long, and A. L. Selman. Quantitative relativizations of complexity classes. *SIAM Journal on Computing*, 13:461–487, 1984.

[Cai87] J. Cai. Probability one separation of the Boolean hierarchy. In *4th Annual Symposium on Theoretical Aspects of Computer Science*, volume 247 of *Lecture Notes in Computer Science*, pages 148–158. Springer-Verlag, 1987.

[CG93] 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.

[CGH+88] 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.

[CGH+89] 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.

[CGL97] R. Chang, W. I. Gasarch, and C. Lund. On bounded queries and approximation. *SIAM Journal on Computing*, 26(1):188–209, February 1997.

[Cha94] R. Chang. Structural complexity column: A machine model for NP-approximation problems and the revenge of the Boolean hierarchy. *Bulletin of the European Association for Theoretical Computer Science*, 54:166–182, October 1994.

[Cha96] R. Chang. On the query complexity of clique size and maximum satisfiability. *Journal of Computer and System Sciences*, 53(2):298–313, October 1996.

[CK96] 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.

[CKST95] P. Crescenzi, V. Kann, R. Silvestri, and L. Trevisan. Structure in approximation classes. In *Proceedings of the 1st Computing and Combinatorics Conference*, volume 959 of *Lecture Notes in Computer Science*, pages 539–548. Springer-Verlag, August 1995.

[CT94] P. Crescenzi and L. Trevisan. On approximation scheme preserving reducibility and its applications. In *Proceedings of the 14th Conference on the Foundations of Software Technology and Theoretical Computer Science*, volume 880 of *Lecture Notes in Computer Science*, pages 330–341. Springer-Verlag, December 1994.

[CW] A. Cohen and A. Wigderson. Dispersers, deterministic amplification, and weak random sources. Full version of [CW89] available on-line `http://www.cs.huji.ac.il/~avi/`.

[CW89] A. Cohen and A. Wigderson. Dispersers, deterministic amplification, and weak random sources. In *Proceedings of the IEEE Symposium on Foundations of Computer Science*, pages 14–19, 1989.

[FGL+91] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy. Approximating clique is almost NP-complete. In *Proceedings of the IEEE Symposium on Foundations of Computer Science*, pages 2–12, 1991.

[Hem89] L. Hemachandra. The strong exponential hierarchy collapses. *Journal of Computer and System Sciences*, 39(3):299–322, 1989.

[HHH96] E. Hemaspaandra, L. A. Hemaspaandra, and H. Hempel. Downward translation in the polynomial hierarchy. Technical Report TR-630, University of Rochester, Department of Computer Science, July 1996. Also appeared in *Proceedings of the 14th Annual Symposium on Theoretical Aspects of Computer Science*, *Lecture Notes in Computer Science*, Springer-Verlag.

[Kad88] J. Kadin. The polynomial time hierarchy collapses if the Boolean hierarchy collapses. *SIAM Journal on Computing*, 17(6):1263–1282, December 1988.

[KMSV94] S. Khanna, R. Motwani, M. Sudan, and U. Vazirani. On syntactic versus computational views of approximability. In *Proceedings of the IEEE Symposium on Foundations of Computer Science*, pages 819–830, November 1994.

[Kre88] M. W. Krentel. The complexity of optimization problems. *Journal of Computer and System Sciences*, 36(3):490–509, 1988.

[KT94] J. Köbler and T. Thierauf. Complexity-restricted advice functions. *SIAM Journal on Computing*, 23(2):261–275, April 1994.

[Lon85] T. J. Long. On restricting the size of oracles compared with restricting access to oracles. *SIAM Journal on Computing*, 14:585–597, 1985.

[Lov75] L. Lovász. On the ratio of optimal integral and fractional covers. *Discrete Mathematics*, 13:383–390, 1975.

[Pap94] C. H. Papadimitriou. *Compuational Complexity.* Addison-Wesley, 1994.

[PY82] C. H. Papadimitriou and M. Yannakakis. The complexity of facets (and some facets of complexity). In *ACM Symposium on Theory of Computing*, pages 255–260, 1982.

[Wag88] K. Wagner. Bounded query computations. In *Proceedings of the 3rd Structure in Complexity Theory Conference*, pages 260–277, June 1988.

[Wag90] K. Wagner. Bounded query classes. *SIAM Journal on Computing*, 19:833–846, 1990.

[Yap83] C. Yap. Some consequences of non-uniform conditions on uniform classes. *Theoretical Computer Science*, 26(3):287–300, 1983.