

One Bit of Advice

Harry Buhrman¹, Richard Chang^{2*}, and Lance Fortnow³

¹ CWI & University of Amsterdam. Address: CWI, INS4, P.O. Box 94709, Amsterdam, The Netherlands. buhrman@cwi.nl.

² Department of Computer Science and Electrical Engineering, University of Maryland Baltimore County, 1000 Hilltop Circle, Baltimore, MD 21250, USA. chang@umbc.edu.

³ NEC Laboratories America, 4 Independence Way, Princeton, NJ 08540, USA. fortnow@nec-labs.com

Abstract. The results in this paper show that coNP is contained in NP with 1 bit of advice (denoted $\text{NP}/1$) if and only if the Polynomial Hierarchy (PH) collapses to D^{P} , the second level of the Boolean Hierarchy (BH). Previous work showed that $\text{BH} \subseteq \text{D}^{\text{P}} \implies \text{coNP} \subseteq \text{NP}/\text{poly}$. The stronger assumption that $\text{PH} \subseteq \text{D}^{\text{P}}$ in the new result allows the length of the advice function to be reduced to a single bit and also makes the converse true. The one-bit case can be generalized to any constant k :

$$\text{PH} \subseteq \text{BH}_{2^k} \iff \text{coNP} \subseteq \text{NP}/k$$

where BH_{2^k} denotes the 2^k -th level of BH and NP/k denotes the class NP with k -bit advice functions.

1 Introduction

The results in this paper are motivated in part by the search for a total upward collapse of the Polynomial Hierarchy (PH) under the assumption that one query to NP is just as powerful as two queries — i.e., the assumption that $\text{P}^{\text{NP}[1]} = \text{P}_{\text{tt}}^{\text{NP}[2]}$. Kadin was first to show that if $\text{P}^{\text{NP}[1]} = \text{P}_{\text{tt}}^{\text{NP}[2]}$ then the PH collapses to Σ_3^{P} . Chang and Kadin improved the collapse of PH to the Boolean Hierarchy over Σ_2^{P} [12]. This was further improved by Beigel, Chang and Ogihara to a class just above Σ_2^{P} [3]. Most recently Fortnow, Pavan and Sengupta, building on Buhrman and Fortnow [4], pushed the collapse below the Σ_2^{P} level and showed that $\text{P}^{\text{NP}[1]} = \text{P}_{\text{tt}}^{\text{NP}[2]}$ implies that $\text{PH} \subseteq \text{S}_2^{\text{P}}$ [13]. Separately, Chang and Kadin noted that $\text{P}^{\text{NP}[1]} = \text{P}_{\text{tt}}^{\text{NP}[2]}$ implies that $\text{P}^{\text{NP}[O(\log n)]} \subseteq \text{P}^{\text{NP}[1]}$ [11]. This was further improved by Buhrman and Fortnow to $\text{P}^{\text{NP}} \subseteq \text{P}^{\text{NP}[1]}$ [4]. Since $\text{S}_2^{\text{P}} \subseteq \text{ZPP}^{\text{NP}}$ [7], we have the following situation:

$$\text{P}^{\text{NP}[1]} = \text{P}_{\text{tt}}^{\text{NP}[2]} \implies \text{PH} \subseteq \text{ZPP}^{\text{NP}}.$$

$$\text{P}^{\text{NP}[1]} = \text{P}_{\text{tt}}^{\text{NP}[2]} \implies \text{P}^{\text{NP}} \subseteq \text{P}^{\text{NP}[1]}.$$

* Supported in part by the University of Maryland Institute for Advanced Computer Studies.

This is almost a complete upward collapse of PH down to $P^{NP[1]}$ except for the “gap” between P^{NP} and ZPP^{NP} . Closing this gap might be done with a proof that $P^{NP[1]} = P_{tt}^{NP[2]} \implies ZPP^{NP} \subseteq P^{NP}$. However, the possibility remains for less direct approaches.

The question we ask in this paper is: under what conditions could we get a total collapse of PH below $P_{tt}^{NP[2]}$? We show that

$$PH \subseteq D^P \iff coNP \subseteq NP/1.$$

Here, the NP/1 is NP with one bit of advice and the class D^P consists of those languages that can be expressed as the difference of two NP languages. Note that $P^{NP[1]} \subseteq D^P \subseteq P_{tt}^{NP[2]}$ and that the proofs of most of the upward collapse results involving $P^{NP[1]}$ and $P_{tt}^{NP[2]}$ actually start with the argument that $P^{NP[1]} = P_{tt}^{NP[2]}$ implies that D^P is closed under complementation. Previous results have shown that under the weaker assumption that the Boolean Hierarchy collapses to D^P , $coNP \subseteq NP/poly$ [16]. In contrast, the results in this paper make the stronger assumption that PH collapses to D^P . The stronger assumption allows us to reduce the advice to just one bit and also allows us to prove the converse. Our results also generalize to the k -bit case. We are able to show that:

$$PH \subseteq BH_{2^k} \iff coNP \subseteq NP/k.$$

2 Preliminaries

We use the standard definition and notation for complexity classes with advice (a.k.a., non-uniform complexity) [17]. An important consideration in the definition below is that the advice function depends only on the length of x and not on x itself.

Definition 1. Let L be a language and $f : \mathbb{N} \rightarrow \{0, 1\}^*$ be an *advice function*. Then, we define $L/f = \{x \mid \langle x, f(|x|) \rangle \in L\}$. For a complexity class \mathcal{C} and a class of functions \mathcal{F} , $\mathcal{C}/\mathcal{F} = \{L/f \mid L \in \mathcal{C}, f \in \mathcal{F}\}$. Thus, NP/poly denotes the class of languages recognized by NP machines with advice functions f where $|f(n)|$ is bounded by a polynomial in n . For this paper, we will consider the classes NP/1 and NP/ k where the NP machines have, respectively, one-bit and k -bit advice functions (i.e., $|f(n)| = 1$ and $|f(n)| = k$).

The Boolean Hierarchy is a generalization of the class D^P defined by Papadimitriou and Yannakakis [20]. For constant k , the k th level of the Boolean Hierarchy can be defined simply as nested differences of NP languages [5, 6].

Definition 2. Starting with $BH_1 = NP$ and $BL_1 = SAT$, we define the k th level of the Boolean Hierarchy and its complete languages by:

$$\begin{aligned} BH_{k+1} &= \{ L_1 - L_2 \mid L_1 \in NP \text{ and } L_2 \in BH_k \} \\ coBH_k &= \{ L \mid \bar{L} \in BH_k \} \end{aligned}$$

$$\begin{aligned}
\text{BL}_{2k} &= \{ \langle x_1, \dots, x_{2k} \rangle \mid \langle x_1, \dots, x_{2k-1} \rangle \in \text{BL}_{2k-1} \text{ and } x_{2k} \in \overline{\text{SAT}} \} \\
\text{BL}_{2k+1} &= \{ \langle x_1, \dots, x_{2k+1} \rangle \mid \langle x_1, \dots, x_{2k} \rangle \in \text{BL}_{2k} \text{ or } x_{2k+1} \in \text{SAT} \} \\
\text{coBL}_k &= \{ \langle x_1, \dots, x_k \rangle \mid \langle x_1, \dots, x_k \rangle \notin \text{BL}_k \}.
\end{aligned}$$

Thus, BL_k is \leq_m^P -complete for BH_k [5, 6]. We let $\text{BH} = \bigcup_{k=1}^{\infty} \text{BH}_k$. Also, for historical convention, we let $\text{D}^P = \text{BH}_2$, $\text{co-D}^P = \text{coBH}_2$, $\text{SAT} \wedge \overline{\text{SAT}} = \text{BL}_2$ and $\overline{\text{SAT}} \vee \text{SAT} = \text{coBL}_2$.

The complexity of the Boolean Hierarchy is closely related to the complexity of the bounded query classes which we now define.

Definition 3. Let $q(n)$ be a polynomial-time computable function. We use $\text{P}^{\text{SAT}[q(n)]}$ to denote the set of languages recognized by deterministic polynomial-time Turing machines which on inputs of length n ask at most $q(n)$ queries to SAT, the canonical \leq_m^P -complete language for NP. When the queries are made in parallel, we use the notation $\text{P}_{\text{tt}}^{\text{SAT}[q(n)]}$. We will use P^{SAT} and $\text{P}_{\text{tt}}^{\text{SAT}}$ when the machines are allowed polynomial many queries.

The connection between the Boolean Hierarchy and bounded queries to SAT is rich and varied. We ask the reader to consult the literature for a full accounting [1–6, 8–10, 12, 14–16, 18, 21–23]. For this paper, we make use of the following facts about the Boolean Hierarchy and bounded queries to SAT.

$$\text{P}_{\text{tt}}^{\text{SAT}[k-1]} \subseteq \text{BH}_k \cap \text{coBH}_k \subseteq \text{BH}_k \cup \text{coBH}_k \subseteq \text{P}_{\text{tt}}^{\text{SAT}[k]} \quad [2, 18].$$

$$\text{P}^{\text{SAT}[k]} = \text{P}_{\text{tt}}^{\text{SAT}[2^k-1]} \quad [2, 22, 23].$$

$$\text{BH}_k = \text{coBH}_k \implies \text{BH} = \text{BH}_k \quad [5, 6].$$

$$\text{BH}_k = \text{coBH}_k \implies \overline{\text{SAT}} \in \text{NP/poly} \quad [16].$$

3 Proof of Main Theorem

In this section we will prove the main result, the 1-bit case. The proof for the general case is deferred to the next section. We prove the main result in two parts, one for each direction of the if and only if.

Theorem 1. $\text{coNP} \subseteq \text{NP}/1 \implies \text{PH} \subseteq \text{D}^P$.

Proof: We prove this direction in two steps:

$$\text{coNP} \subseteq \text{NP}/1 \implies \Sigma_2^P \subseteq \text{P}^{\text{SAT}} \quad (1)$$

$$\text{coNP} \subseteq \text{NP}/1 \implies \text{P}^{\text{SAT}} \subseteq \text{D}^P. \quad (2)$$

To prove (1), let U be a \leq_m^P -complete language for Σ_2^P with the usual padding properties and which can be written as:

$$U = \{ \langle x, y \rangle \mid (\exists^P y') (\forall^P z) [y' \leq y \wedge R(x, y', z)] \}$$

for some polynomial-time computable relation R . Since $\text{coNP} \in \text{NP}/1$ by assumption, we can construct an $\text{NP}/1$ machine N_U that recognizes U using standard oracle replacement techniques. We only need to note that by padding, all the oracles queries we replace have the same length. Thus, only one bit of advice is needed for the entire computation.

Next, we construct a P^{SAT} machine D_U which recognizes U without any advice bits. On input $\langle x, y \rangle$, D_U looks for y'_{\max} , the largest $y' \leq y$ such that $(\forall z)[R(x, y', z)]$. D_U finds y'_{\max} using binary search and queries to N_U , which can be answered by SAT if D_U had the advice bit for N_U . (The same advice bit can be used for all the queries to N_U during one binary search.) Since D_U does not have the advice bit, it simply tries both 0 and 1. Let y'_0 and y'_1 be the two values produced by the two trials. Then

$$\langle x, y \rangle \in U \iff (\forall^P z)[R(x, y'_0, z)] \vee (\forall^P z)[R(x, y'_1, z)]. \quad (3)$$

D_u can verify the right hand side of (3) with its SAT oracle. Thus, $\Sigma_2^P \subseteq \text{P}^{\text{SAT}}$ and we have established (1).

To prove that (2) also holds, we show that $\text{coNP} \subseteq \text{NP}/1$ implies that LEXMAXSAT , defined below, is in D^P .

$$\begin{aligned} \text{LEXMAXSAT} = \\ \{ \varphi \mid \text{the lexically largest satisfying assignment of } \varphi \text{ ends with } 1 \}. \end{aligned}$$

Since LEXMAXSAT is \leq_m^P -complete for P^{SAT} [19], we have $\text{P}^{\text{SAT}} \subseteq \text{D}^P$. Note that $\Sigma_2^P \subseteq \text{P}^{\text{SAT}} \implies \text{PH} \subseteq \text{P}^{\text{SAT}}$. Thus by (1) we have $\text{PH} \subseteq \text{D}^P$.

Using the assumption that $\text{coNP} \subseteq \text{NP}/1$, we can construct an $\text{NP}/1$ machine N_{LMS} that given φ outputs α_{\max} , the lexically largest satisfying assignment for φ . When N_{LMS} is given the correct advice bit, all of its computation paths that produce an output (henceforth, the output paths) will output α_{\max} . If N_{LMS} has the wrong advice bit, it might output different values on different output paths or it might not have any output paths. We program N_{LMS} to explicitly check that every string it outputs is at least a satisfying assignment of φ . This will be useful below when we need to consider the behavior of N_{LMS} given the wrong advice.

We define two NP languages A_1 and A_2 and claim that $\text{LEXMAXSAT} = A_1 - A_2$. First, we let

$$A_1 = \{ \varphi \mid N_{\text{LMS}}(\varphi, 0) \text{ or } N_{\text{LMS}}(\varphi, 1) \text{ outputs a value that ends with } 1 \}.$$

Recall that in our notation $N_{\text{LMS}}(\varphi, 0)$ and $N_{\text{LMS}}(\varphi, 1)$ represents the computations of N_{LMS} given advice bit 0 and 1, respectively.

The language A_2 is defined by an NP machine N_{A_2} . On input φ , N_{A_2} looks for a computation path of $N_{\text{LMS}}(\varphi, 0)$ and a computation path of $N_{\text{LMS}}(\varphi, 1)$ that output different satisfying assignments for φ . Call these assignments α_1 and α_2 and w.o.l.o.g. assume that $\alpha_1 < \alpha_2$. $N_{A_2}(\varphi)$ accepts if α_1 ends with a 1 and α_2 ends with a 0.

Clearly, A_1 and A_2 are NP languages. To see that $\text{LEXMAXSAT} = A_1 - A_2$, first suppose that $\varphi \in \text{LEXMAXSAT}$. Since one of $N_{\text{LMS}}(\varphi, 0)$ and $N_{\text{LMS}}(\varphi, 1)$

has the correct advice bit, one of them must output α_{\max} . Since α_{\max} ends with 1, $\varphi \in A_1$. On the other hand, φ cannot be in A_2 by maximality of α_{\max} . Thus, $\varphi \in \text{LEXMAXSAT} \implies \varphi \in A_1 - A_2$.

Conversely, suppose that $\varphi \notin \text{LEXMAXSAT}$. Then, the largest satisfying assignment ends with a 0. So, the computation with the correct advice bit will never output a value ending with a 1. Thus, $\varphi \in A_1$ only in the case that the computation with the wrong advice bit outputs a value $\alpha < \alpha_{\max}$ and α ends with a 1. However, in this case, φ is also in A_2 . Thus, $\varphi \notin \text{LEXMAXSAT} \implies \varphi \notin A_1 - A_2$. \square

In the next theorem, we show that $\text{PH} \subseteq \text{D}^{\text{P}} \implies \text{coNP} \subseteq \text{NP}/1$ using the hard/easy argument which was used to show that $\text{D}^{\text{P}} = \text{co-D}^{\text{P}}$ implies a collapse of PH [16]. Suppose that $\text{D}^{\text{P}} = \text{co-D}^{\text{P}}$. Then $\text{SAT} \wedge \overline{\text{SAT}} \leq_m^{\text{P}} \overline{\text{SAT}} \vee \text{SAT}$ via some polynomial-time reduction h . Using the reduction h , we define a *hard string*:

Definition 4. Suppose $\text{SAT} \wedge \overline{\text{SAT}} \leq_m^{\text{P}} \overline{\text{SAT}} \vee \text{SAT}$ via some polynomial-time reduction h . Then, a string H is called a *hard string for length n* , if $|H| = n$, $H \in \overline{\text{SAT}}$ and for all x , $|x| = n$, $\langle x, H \rangle \xrightarrow{h} \langle G_1, G_2 \rangle$ with $G_2 \notin \text{SAT}$. If $F \in \overline{\text{SAT}}$, $|F| = n$ and F is not a hard string for length n , then we say that F is an *easy string*.

Suppose that we were given a hard string H for length n . Then the NP procedure below accepts a formula F of length n if and only if $F \in \overline{\text{SAT}}$.

PROCEDURE Hard(F): Compute $h(F, H) = \langle G_1, G_2 \rangle$. Guess a truth assignment α to the variables in G_1 . Accept if α satisfies G_1 .

On the other hand, if there are no hard strings for length n — i.e., all formulas in $\overline{\text{SAT}}^{-n}$ are easy — we also have an NP procedure for $\overline{\text{SAT}}^{-n}$.

PROCEDURE Easy(F): Guess a string x with $|x| = |F|$. Compute $h(x, F) = \langle G_1, G_2 \rangle$. Guess a truth assignment α to the variables of G_2 . Accept if α satisfies G_2 .

The correctness of Procedures Hard and Easy follows directly from the definitions of $\text{SAT} \wedge \overline{\text{SAT}}$, $\overline{\text{SAT}} \vee \text{SAT}$ and hard strings [16]. Since a polynomial advice function can provide an NP machine with a hard string for each length n or with the advice that all strings in $\overline{\text{SAT}}^{-n}$ are easy, $\text{D}^{\text{P}} = \text{co-D}^{\text{P}} \implies \text{coNP} \subseteq \text{NP}/\text{poly}$. For this paper, we want to show that $\text{PH} \subseteq \text{D}^{\text{P}} \implies \text{coNP} \subseteq \text{NP}/1$ which is both a stronger hypothesis and a strong consequence. Hence, we need to exploit the assumption that $\text{PH} \subseteq \text{D}^{\text{P}}$.

Theorem 2. $\text{PH} \subseteq \text{D}^{\text{P}} \implies \text{coNP} \subseteq \text{NP}/1$.

Proof: Suppose that $\text{PH} \subseteq \text{D}^{\text{P}}$. Then, $\text{D}^{\text{P}} = \text{co-D}^{\text{P}}$ via some \leq_m^{P} -reduction h . Now, fix a length n and consider only inputs strings φ of length n . Our goal is to find a hard string for length n or determine that there are no hard strings

for length n . Then we can use Procedure Hard or Easy to accept if and only if $\varphi \in \overline{\text{SAT}}$.

Note that the lexically smallest hard string for length n can be found by a $\text{P}^{\text{NP}^{\text{NP}}}$ machine, because the set of hard strings is in coNP . Since $\text{PH} \subseteq \text{D}^{\text{P}}$, the language HARDBITS defined below is also in D^{P} .

$$\begin{aligned} \text{HARDBITS} = & \{ \langle 1^n, 0 \rangle \mid \text{there are no hard strings for length } n \} \\ & \cup \{ \langle 1^n, i \rangle \mid \text{the } i\text{th bit of the lexically smallest hard string} \\ & \text{for length } n \text{ is } 1 \}. \end{aligned}$$

Since $\text{D}^{\text{P}} \subseteq \text{P}_{\text{tt}}^{\text{SAT}[2]}$, HARDBITS is recognized by some $\text{P}_{\text{tt}}^{\text{SAT}[2]}$ machine M_{HB} . Now, consider the following $n+1$ computations of M_{HB} : $M_{\text{HB}}(1^n, 0)$, $M_{\text{HB}}(1^n, 1)$, $M_{\text{HB}}(1^n, 2)$, \dots , $M_{\text{HB}}(1^n, n)$. If we are given the accept/reject results of all $n+1$ computations, then we can recover the lexically smallest hard string for length n or conclude that there are no hard strings for length n . Let W be the set of oracle queries made to SAT in these $n+1$ computations. Without loss of generality we assume that the queries have the same length m . In the remainder of the proof we construct an $\text{NP}/1$ machine that can determine the satisfiability of the formulas in W . The one-bit of advice for our $\text{NP}/1$ computation is 0 if all the strings in $W \cap \overline{\text{SAT}}$ are easy and 1 if W contains at least one hard string for length m . Note that the set W depends only on $|\varphi|$ and not on φ itself, so the one bit of advice is indeed the same for all inputs of length n . Our $\text{NP}/1$ computation is divided into two cases, depending on the advice. Putting the two cases together gives us $\text{coNP} \subseteq \text{NP}/1$.

Case 1: all strings in $W \cap \overline{\text{SAT}}$ are easy. We construct an NP machine N_e that accepts if and only if the original input φ of length n is unsatisfiable. N_e first constructs the set W by simulating M_{HB} . Then, for each string w in W , N_e either guesses a satisfying assignment for w or uses Procedure Easy to verify that w is unsatisfiable. The only computation branches of N_e that survive this step are the ones that have correctly guessed the satisfiability of each $w \in W$.

Next, N_e simulates each of the $n+1$ computations of M_{HB} for HARDBITS . Since N_e has the answers to each oracle query, the simulations can be completed and N_e can reconstruct the lexically smallest hard string for length n or determine that there are no hard strings for length n . Then N_e uses either Procedure Hard or Easy and accepts the original input φ if and only if $\varphi \in \overline{\text{SAT}}$.

Case 2: W contains at least one hard string. Our advantage in this case is that we can look for a hard string for length m just among the $\leq 2n+2$ strings in W instead of all 2^m strings of length m . We construct an NP machine N_h which nondeterministically places each string $w \in W$ into three sets: W_{SAT} , W_{easy} and W_{hard} . The intention is that W_{SAT} has all the strings in $W \cap \text{SAT}$, W_{easy} has all the easy strings in $W \cap \overline{\text{SAT}}$ and W_{hard} has the remaining strings, the hard strings in $W \cap \overline{\text{SAT}}$. As in the previous case, N_h can verify that the strings in W_{SAT} are satisfiable and that the strings in W_{easy} are easy. However, N_h will not be

able to verify that the strings in W_{hard} are unsatisfiable and hard. Fortunately, we only need to know that the strings in W_{hard} are unsatisfiable. That would be enough to simulate the $n + 1$ computations of M_{HB} for **HARDBITS**.

To check that $W_{\text{hard}} \subseteq \overline{\text{SAT}}$, N_h takes each string x in W_{hard} , assumes for the moment that x is indeed hard and uses x to check that every string w in W_{hard} is unsatisfiable. That is, for each pair of strings w, x in W_{hard} , N_h computes $h(w, x) = \langle G_1, G_2 \rangle$ and guesses a satisfying assignment for G_1 . If N_h succeeds for every pair (w, x) then we say that W_{hard} has been verified.

Now, suppose that some computation branch of N_h has verified W_{SAT} , W_{easy} and W_{hard} . Since W contains at least one hard string z , N_h must have placed z in W_{hard} . Then z would have been used to test that every $w \in W_{\text{hard}}$ is indeed unsatisfiable. Since z really is a hard string, we are assured that $W_{\text{hard}} \subseteq \overline{\text{SAT}}$. Thus, we can claim that $W_{\text{SAT}} = W \cap \text{SAT}$. Furthermore, some computation branch of N_h guessed W_{SAT} , W_{easy} and W_{hard} to be exactly the satisfiable, easy and hard strings in W . Therefore, at least one computation branch of N_h has verified its W_{SAT} , W_{easy} and W_{hard} and has determined the satisfiability of every string in W .

As in the previous case, since N_h knows the answer to every oracle query in the $n + 1$ computations of M_{HB} for **HARDBITS**, N_h can recover the lexicographically smallest hard string for length n and use it to nondeterministically recognize the unsatisfiability of the original input φ . \square

4 Generalizations

In this section we generalize the main theorem and show that $\text{PH} \subseteq \text{BH}_{2^k} \iff \text{coNP} \subseteq \text{NP}/k$. Recall that BH_{2^k} is the 2^k th level of the Boolean Hierarchy and that $\text{D}^{\text{P}} = \text{BH}_2$, so the preceding theorems are special cases of the ones in this section. As before, we prove each direction separately:

Theorem 3. $\text{coNP} \subseteq \text{NP}/k \implies \text{PH} \subseteq \text{BH}_{2^k}$.

Proof: The first step of the proof of Theorem 1 showed that $\text{coNP} \subseteq \text{NP}/1 \implies \text{PH} \subseteq \text{P}^{\text{SAT}}$. This step generalizes to k bits of advice in a straightforward manner. The P^{SAT} machine D_U for the Σ_2^{P} -complete language U simply has to try all 2^k possible advice strings. D_U on input $\langle x, y \rangle$ obtains 2^k candidates y'_1, \dots, y'_{2^k} for y'_{max} . For each y'_i , it checks whether $(\forall^{\text{P}} z)[R(x, y'_i, z)]$ using its NP oracle. Then, $\langle x, y \rangle \in U$ if and only if $(\forall^{\text{P}} z)[R(x, y'_i, z)]$ for some i , $1 \leq i \leq 2^k$.

Next, we show that $\text{LEXMAXSAT} \in \text{BH}_{2^k}$ which completes the proof, since LEXMAXSAT is \leq_m^{P} -complete for P^{SAT} . As in Theorem 1, we use an NP/k machine N_{LMS} which, given the right advice, outputs the largest satisfying assignment α_{max} of its input formula φ . We will consider 2^k computation trees of N_{LMS} on input φ denoted $N_{\text{LMS}}(\varphi, 0^k), \dots, N_{\text{LMS}}(\varphi, 1^k)$ (one for each k -bit advice string).

Given the correct advice, N_{LMS} will output α_{max} on all of its output paths. Given the wrong advice, N_{LMS} might output one or more incorrect values or have no output paths. Recall that we had previously rigged N_{LMS} so that it only

outputs satisfying assignments of φ even when it is given the wrong advice. Our objective is to construct 2^k NP languages A_1, \dots, A_{2^k} such that

$$\varphi \in \text{LEXMAXSAT} \iff \varphi \in A_1 - (A_2 - (\dots - A_{2^k}) \dots).$$

We use the mind-change technique which was used to show that $\text{P}^{\text{SAT}[k]} = \text{P}_{\text{tt}}^{\text{SAT}[2^k-1]}$ [2, 22, 23]. We construct 2^k NP machines $N_{A_1}, \dots, N_{A_{2^k}}$. On input φ , N_{A_i} does the following:

PROCEDURE $A_i(\varphi)$

1. Guess i different advice strings $\sigma_1, \dots, \sigma_i \in \{0, 1\}^k$ in any possible order.
2. For each j , $1 \leq j \leq i$, guess a computation path of $N_{\text{LMS}}(\varphi, \sigma_j)$ that produces an output. Call this output string α_j .
3. Verify that $\alpha_1 < \alpha_2 < \dots < \alpha_i$ in lexical ordering.
4. Verify that the last bit of α_1 is a 1 and that for each j , $1 \leq j < i$, the last bit of α_j is different from the last bit of α_{j+1} .
5. Accept if Steps 2, 3 and 4 succeed.

When N_{A_i} accepts, we think of $\alpha_1 < \alpha_2 < \dots < \alpha_i$ as a sequence of mind changes. Now suppose that the longest sequence of mind changes is $\alpha_1 < \alpha_2 < \dots < \alpha_\ell$. Then, the last bit of α_ℓ must be the same as the last bit of α_{max} . This is true if $\ell = 2^k$, since then all advice strings in $\{0, 1\}^k$ were used in Step 2, including the correct advice which produces α_{max} . Since every α_j is satisfiable, α_{max} must equal α_ℓ . If $\ell \neq 2^k$ then appending α_{max} to the end of the sequence would create a longer sequence of mind changes contradicting the maximality of ℓ . Appending α_{max} would be possible since N_{LMS} outputs α_{max} given the correct advice.

Let A_i be the set of formulas φ accepted by N_{A_i} and let $A = A_1 - (A_2 - (\dots - A_{2^k}) \dots)$. We claim that $\varphi \in \text{LEXMAXSAT}$ if and only if $\varphi \in A$. Let ℓ be the largest i such that $\varphi \in A_i$ or 0 if no such i exists. Note that $\varphi \in A$ if and only if ℓ is odd because $A_1 \supseteq A_2 \supseteq \dots \supseteq A_{2^k}$. Now suppose that $\varphi \in \text{LEXMAXSAT}$. Then, the last bit of α_{max} is 1, so $\varphi \in A_1$ and $\ell \geq 1$. Let $\alpha_1 < \alpha_2 < \dots < \alpha_\ell$ be a mind change sequence found by N_{A_ℓ} on input φ . As we argued above, the last bit of α_ℓ must be the same as the last bit of α_{max} which is a 1. Since α_1 ends with 1 and the α_j 's must alternate between ending with 1 and ending with 0, ℓ must be odd. Thus, $\varphi \in A$.

Conversely, suppose that $\varphi \in A$. Then, ℓ must be odd. Again, looking at the mind change sequence, $\alpha_1 < \alpha_2 < \dots < \alpha_\ell$, we conclude that α_ℓ must end with 1 and thus α_{max} must end with 1. Therefore, $\varphi \in \text{LEXMAXSAT}$. \square

In the next proof, we extend the hard/easy argument in the previous section to show that $\text{PH} \subseteq \text{BH}_{2^k} \implies \text{coNP} \subseteq \text{NP}/k$. A key element of the proof is the generalization of a hard string to a hard sequence [12].

Definition 5. For $\vec{y} = \langle y_1, \dots, y_s \rangle$, let $\vec{y}^R = \langle y_s, \dots, y_1 \rangle$ be the reversal of the sequence. Let π_i and $\pi_{i,j}$ be the projection functions such that $\pi_i(\vec{y}) = y_i$ and $\pi_{i,j}(\vec{y}) = \langle y_i, \dots, y_j \rangle$.

Definition 6. Suppose that $\text{BL}_r \leq_m^{\text{P}} \text{coBL}_r$ via a polynomial-time reduction h . For $0 \leq s \leq r - 1$, let $\ell = r - s$. Then, $\vec{x} = \langle x_1, \dots, x_s \rangle$ is a *hard sequence for length n with respect to h* , if the following hold:

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

We refer to s 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. Furthermore, given a hard sequence \vec{x} , we say that a string w is *easy* with respect to \vec{x} if $|w| = n$ and there exists $u_1, \dots, u_{\ell-1} \in \{0, 1\}^n$ such that $\pi_\ell(h(u_1, \dots, u_{\ell-1}, w, \vec{x}^R)) \in \text{SAT}$. We say that a hard sequence \vec{x} is a *maximal hard sequence*, if for all $w \in \{0, 1\}^n$, $\langle x_1, \dots, x_s, w \rangle$ is not a hard sequence. A *maximum* hard sequence is a hard sequence with maximum order among all the hard sequences for length n .

As with hard strings, a hard sequence \vec{x} allows an NP machine to verify that a string w is unsatisfiable when w is easy with respect to \vec{x} . It follows directly from the definitions of BL_r and coBL_r , that if \vec{x} is a hard sequence and $\pi_\ell(h(u_1, \dots, u_{\ell-1}, w, \vec{x}^R)) \in \text{SAT}$ for any $u_1, \dots, u_{\ell-1} \in \{0, 1\}^n$, then w must be unsatisfiable [12]. Since every string of length n in $\overline{\text{SAT}}$ must be easy with respect to a maximum hard sequence for length n , finding a maximum hard sequence will allow us to recognize $\overline{\text{SAT}}^n$ with an NP machine.

Theorem 4. $\text{PH} \subseteq \text{BH}_{2^k} \implies \text{coNP} \subseteq \text{NP}/k$.

Proof: Suppose that $\text{PH} \subseteq \text{BH}_{2^k}$. Then, $\text{BL}_{2^k} \leq_m^{\text{P}} \text{coBL}_{2^k}$ via some polynomial-time reduction h . Fix a length n and consider only input strings φ of length n . Our goal is to find a maximum hard sequence \vec{x} for length n using an NP machine with k bits of advice. Since φ must be easy with respect to \vec{x} , we get an NP procedure that accepts if and only if $\varphi \in \overline{\text{SAT}}$.

Since the set of hard sequences is in coNP , a $\text{P}^{\text{NP}^{\text{NP}}}$ machine can use binary search to find the lexicographically smallest maximum hard sequence for length n . Moreover, since $\text{PH} \subseteq \text{BH}_{2^k}$, the language HARDBITS defined below can be recognized by a $\text{P}_{\text{tt}}^{\text{SAT}[2^k]}$ machine M_{HB} .

$$\begin{aligned} \text{HARDBITS} = & \{ \langle 1^n, 0 \rangle \mid \text{the maximum hard sequence for length } n \text{ has order } 0 \} \\ & \cup \{ \langle 1^n, i \rangle \mid \text{the } i\text{th bit of the lexicographically smallest maximum} \\ & \text{hard sequence for length } n \text{ is } 1 \}. \end{aligned}$$

Running M_{HB} on $2^k n + 1$ input strings will allow us to recover a maximum hard sequence for length n . As before, we assume that all the queries made by M_{HB} in these computations have a fixed length m . Let W be the set of these length m queries. There are at most $2^{2^k n} + 2^k$ strings in W .

Let $\text{HARD}(m, W)$ be the set of hard sequences for length m where every component of the hard sequence is a string from W . Since the number of all possible sequences $< 2^k(2^{2^k}n + 2^k)^{2^k}$, $|\text{HARD}(m, W)|$ is polynomially bounded. Furthermore, $\text{HARD}(m, W)$ depends only on $|\varphi|$ and not on φ itself. Thus, we can define a k -bit advice function that provides the maximum order of the hard sequences in $\text{HARD}(m, W)$. Call this value z .

We construct an NP/ k machine N for $\overline{\text{SAT}}$ as follows. On input x and given advice z , N first guesses two sets W_{SAT} and H . The set W_{SAT} is a subset of W . If N guesses W_{SAT} correctly, then W_{SAT} would be exactly $W \cap \text{SAT}$. The set H is a set of sequences with $\leq z$ components where each component is a string in W . One correct guess for H is the set $\text{HARD}(m, W)$. There may be other correct guesses for H .

N verifies W_{SAT} and H as follows. For each $w \in W_{\text{SAT}}$, N guesses a satisfying assignment for w . It remains possible that some $w \in W - W_{\text{SAT}}$ is satisfiable. Next, we try to verify that each sequence $\vec{y} = \langle y_1, \dots, y_s \rangle \in H$ is a hard sequence. First, each y_i must be an element of $W - W_{\text{SAT}}$, since the components of a hard sequence must be unsatisfiable. Also, for each $\vec{y} = \langle y_1, \dots, y_s \rangle \in H$ and each $w \in W - W_{\text{SAT}}$, 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 EasyTest($\langle y_1, \dots, y_s \rangle, w$)

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

Clearly, if $W_{\text{SAT}} = W \cap \text{SAT}$ and $H = \text{HARD}(m, W)$, then every verification step will succeed. We claim that if W_{SAT} and H pass every verification step, then $W_{\text{SAT}} = W \cap \text{SAT}$. (Note: we do not claim that H must also equal $\text{HARD}(m, W)$.)

Suppose that H passes every verification step. Let $\vec{y} = \langle y_1, \dots, y_s \rangle$ be any hard sequence from $\text{HARD}(m, W)$. We claim that \vec{y} must be in H . Suppose not. W.o.l.o.g. we can assume that the empty sequence is in H . Thus there exists i , $0 \leq i < s$, such that $\langle y_1, \dots, y_i \rangle \in H$ but $\langle y_1, \dots, y_{i+1} \rangle \notin H$. Then, y_{i+1} should be easy with respect to the hard sequence $\langle y_1, \dots, y_i \rangle$. This will prompt N to run the EasyTest procedure on $\langle y_1, \dots, y_i \rangle$ and y_{i+1} . However, $\langle y_1, \dots, y_{i+1} \rangle$ is in reality a hard sequence, so EasyTest($\langle y_1, \dots, y_i \rangle, y_{i+1}$) will fail. Thus, H would not have passed every verification, which is a contradiction. Therefore, $\text{HARD}(m, W) \subseteq H$.

Next, we claim that $W_{\text{SAT}} = W \cap \text{SAT}$. Fix a string $w \in W - W_{\text{SAT}}$ and let \vec{x} be a hard sequence in H of order z (the maximum order given by the advice function). We know that such a hard sequence exists since z was given by the advice function and we have just shown that $\text{HARD}(m, W) \subseteq H$. Since $\langle \vec{x}, w \rangle \notin H$, N must have succeeded in the procedure call EasyTest(\vec{x}, w). Then, there exists $u_1, \dots, u_{\ell-1} \in \{0, 1\}^m$ such that $\pi_\ell(h(u_1, \dots, u_{\ell-1}, w, \vec{x}^R)) \in \text{SAT}$,

where $\ell = 2^k - z$. By the definitions of BL_{2^k} and coBL_{2^k} , this is enough to imply that $w \in \overline{\text{SAT}}$. Thus, every string $w \in W - W_{\text{SAT}}$ must be unsatisfiable. Since every string in W_{SAT} was already confirmed to be satisfiable, it follows that $W_{\text{SAT}} = W \cap \text{SAT}$.

Finally, some computation path of N will guess the correct W_{SAT} and a correct H which passes every verification step. On such a path, N knows the elements of $W \cap \text{SAT}$. Thus, N can carry out the simulations of M_{HB} and recover the lexically smallest hard sequence for length n . Using this hard sequence, N can then accept the original input φ if and only if $\varphi \in \overline{\text{SAT}}$. Therefore, $\text{coNP} \subseteq \text{NP}/k$. \square

5 Discussion

The results in this paper show a tight connection between the number of bits of advice that an NP machine needs to recognize $\overline{\text{SAT}}$ and the collapse of the Polynomial Hierarchy. On a technical level, this connection is borne out by the mind change technique and the hard/easy argument. We need exactly k bits to encode the order of the maximum hard sequence given a \leq_m^{P} -reduction h from BL_{2^k} to coBL_{2^k} and 2^k mind changes is exactly what we need to recognize a Σ_2^{P} language assuming $\text{coNP} \in \text{NP}/k$. In comparison, Chang and Kadin showed that if $\text{BH} \subseteq \text{BH}_{2^k}$ then an NP^{NP} machine could recognize a Σ_3^{P} -complete language, if it is given the order of the maximum hard sequence as advice [12, Lemma 4.4]. Since this advice can also be encoded in k bits, this previous result showed that $\text{BH} \subseteq \text{BH}_{2^k} \implies \text{PH} \subseteq \text{NP}^{\text{NP}}/k$.

Our new results are obtained not only by strengthening the hypothesis to $\text{PH} \subseteq \text{BH}_{2^k}$ but also through improvements in the hard/easy argument. The technique used by Chang and Kadin required an existential search for a hard sequence (hence requiring an NP^{NP} machine). The current technique involves a search for the hard string or hard sequence in a polynomial sized domain. This technique was first introduced by Hemaspaandra, Hemaspaandra and Hempel [14] and further refined by Buhrman and Fortnow [4] and by Chang [9].

One direction of our results holds true when we consider non-constant advice length. It is fairly easy to extend Theorem 3 to show that $\text{coNP} \subseteq \text{NP}/\log \implies \text{PH} \subseteq \text{P}^{\text{NP}}$. However, the techniques used in Theorem 4 assumes a constant number of queries and cannot be used to show the converse. It remains an open question whether $\text{coNP} \subseteq \text{NP}/\log \iff \text{PH} \subseteq \text{P}^{\text{NP}}$.

References

1. 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.
2. R. Beigel. Bounded queries to SAT and the Boolean hierarchy. *Theoretical Computer Science*, 84(2):199–223, July 1991.

3. 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.
4. H. Buhrman and L. Fortnow. Two queries. *Journal of Computer and System Sciences*, 59(2):182–194, 1999.
5. 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.
6. 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.
7. Jin-Yi Cai. $S_2^P \subseteq ZPP^{NP}$. In *Proceedings of the IEEE Symposium on Foundations of Computer Science*, pages 620–629. IEEE Computer Society, October 2001.
8. R. Chang. On the structure of bounded queries to arbitrary NP sets. *SIAM Journal on Computing*, 21(4):743–754, August 1992.
9. R. Chang. Bounded queries, approximations and the Boolean hierarchy. *Information and Computation*, 169(2):129–159, September 2001.
10. R. Chang, W. I. Gasarch, and C. Lund. On bounded queries and approximation. *SIAM Journal on Computing*, 26(1):188–209, February 1997.
11. R. Chang and J. Kadin. On computing Boolean connectives of characteristic functions. *Mathematical Systems Theory*, 28(3):173–198, May/June 1995.
12. 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.
13. Lance Fortnow, Aduri Pavan, and Samik Sengupta. Proving SAT does not have small circuits with an application to the two queries problem. Technical Report 2002-L014N, NEC Laboratories America Technical Note, 2002.
14. E. Hemaspaandra, L. A. Hemaspaandra, and H. Hempel. Downward collapse within the polynomial hierarchy. *SIAM Journal on Computing*, 28(2):383–393, April 1999.
15. A. Hoene and A. Nickelsen. Counting, selecting, sorting by query-bounded machines. In *Proceedings of the 10th Symposium on Theoretical Aspects of Computer Science*, volume 665 of *Lecture Notes in Computer Science*. Springer-Verlag, 1993.
16. J. Kadin. The polynomial time hierarchy collapses if the Boolean hierarchy collapses. *SIAM Journal on Computing*, 17(6):1263–1282, December 1988.
17. R. Karp and R. Lipton. Turing machines that take advice. *L'Enseignement Mathématique*, 28:191–209, 1982.
18. J. Köbler, U. Schöning, and K. Wagner. The difference and truth-table hierarchies for NP. *RAIRO Theoretical Informatics and Applications*, 21:419–435, 1987.
19. M. W. Krentel. The complexity of optimization problems. *Journal of Computer and System Sciences*, 36(3):490–509, 1988.
20. C. Papadimitriou and M. Yannakakis. The complexity of facets (and some facets of complexity). *Journal of Computer and System Sciences*, 28(2):244–259, April 1984.
21. K. Wagner. Bounded query computations. In *Proceedings of the 3rd Structure in Complexity Theory Conference*, pages 260–277, June 1988.
22. K. Wagner. Bounded query classes. *SIAM Journal on Computing*, 19:833–846, 1990.
23. K. Wagner and G. Wechsung. On the Boolean closure of NP. In *Proceedings of the 1985 International Conference on Fundamentals of Computation Theory*, volume 199 of *Lecture Notes in Computer Science*, pages 485–493. Springer-Verlag, 1985.