Much Ado about Functions

Alan L. Selman*
Department of Computer Science
State University of New York at Buffalo
Buffalo, NY 14260

Abstract

This paper surveys basic results on complexity classes of partial multivalued functions. We stress basic inclusion relations, interesting hierarchies, and results that demonstrate that hierarchies are extant.

1 Introduction

The fundamental data type that a nondeterministic process computes is a *partial multivalued function*, partial because nondeterministic computations do not necessarily accept every input, and multivalued because nondeterministic computations may output different values on different accepting paths. As understanding the power of nondeterminism is one of the fundamental goals of complexity theory, surely, we must study the computational complexity of partial multivalued functions.

The problems that we traditionally think of as set recognition problems are more naturally thought of as functional computational problems. For example, we do not care to know only whether a graph has a hamiltonian, but we want a hamiltonian to be output, if one exists.

It is certainly the case that partial functions are the fundamental objects studied in recursive function theory. So, it is somewhat surprising that complexity theory has largely developed as a study of classification of decision problems, and has somewhat ignored classification of function classes. One reason might be that showing that a problem is complete or hard for a class has in practice been sufficient for showing that no efficient algorithm exists for computing witnesses to the problem. This is because typical combinatorial problems are self-reducible. However, it is not known whether all NP-complete problems in NP are self-reducible. Consequently, it is not known whether the familiar approach works in all cases. By studying complexity classes of partial multivalued functions we address questions about NP search problems and about the difficulty of inverting polynomial-time computable functions. Most importantly, by studying complexity classes of partial multivalued functions, we directly illuminate interesting questions that otherwise would not surface. We will see that properties of complexity classes of partial multivalued functions may be identical to or may differ from those of their corresponding well-known complexity classes of languages. There are several hierarchies of function classes: a query hierarchy that closely reflects the query hierarchy of language classes, a difference hierarchy that only superficially resembles the difference hierarchy for languages, and at least one new hierarchy that seems not to correspond to any collection of language classes. We will see that several of the interesting questions remain open. In brief, we will see that studying the complexity of partial multivalued functions is not *much ado about nothing*.

*E-mail: selman@cs.buffalo.edu. Funding for this research was provided by the National Science Foundation under grant CCR-9400229.

2 Function Classes

To date, most researchers have concentrated on partial multivalued functions that are computed in polynomial time. Notable exceptions are Mocas [Moc93], who has studied partial multivalued functions that are computable in exponential time, and Àlvarez and Jenner [ÀJ93], who have considered functions that are computable by logspace transducers that access oracles in NP. Cai *et al.* [CLL$^+$95] and Ogihara and Regan [OR93] have studied partial multivalued functions that are computed by probabilistic transducers in polynomial time. In this paper, we will confine our attention to questions concerning nondeterministic polynomial time computations.

The definitions to follow origininate for the most part in a paper of Book, Long, and Selman [BLS84]. Results in this section for which we do not give an explicit citation, appear first in a paper by Selman [Sel94].

Fix Σ to be the finite alphabet $\{0,1\}$. Let $f : \Sigma^* \to \Sigma^*$ be a partial multivalued function. We write $f(x) \mapsto y$, if y is a value of f on input string x. Define $graph(f) = \{\langle x,y\rangle \mid f(x) \mapsto y\}$, $dom(f) = \{x \mid \exists y(f(x) \mapsto y)\}$, and

$range(f) = \{y \mid \exists x(f(x) \mapsto y)\}$. If $x \notin dom(f)$, we say that f is undefined at x or that $f(x)$ is undefined.

A transducer T is a nondeterministic Turing machine with a read-only input tape, a write-only output tape, and accepting states in the usual manner. T computes a value y on an input string x if there is an accepting computation of T on x for which y is the final contents of T's output tape. In this case, we will write $T(x) \mapsto y$. Such transducers compute partial, multivalued functions.

- NPMV is the set of all partial, multivalued functions computed by nondeterministic polynomial time-bounded transducers;
- NPSV is the set of all $f \in$ NPMV that are single-valued;
- PF is the set of all partial functions that are computed by deterministic polynomial time-bounded transducers.

Let SAT denote the NP-complete satisfiability problem. The function *sat*, defined by $sat(x) \mapsto y$ if and only if $x \in$ SAT and y is a satisfying assignment of x, is the ubiquitous example of a partial multivalued function; *sat* belongs to NPMV and $dom(sat) = \text{SAT}$.

Example 1 *The following interesting partial functions belong to* NPSV.

(prime factorization) *Let $\{p_i\}$ be the sequence of prime numbers in increasing order. Define $f(n)$, for each positive integer n, to be the finite sequence $\{(p_i, a_i)\}$, $i \geq 0$, such that $n = \prod p_i^{a_i}$. f is single-valued and total because every positive integer has a unique prime factorization. $f \in$* NPSV *because the set of primes belongs to* UP $\cap$ *co*UP *[FK92].*

(discrete logarithm) *Define h so that*

$$\begin{aligned} dom(h) &= \{\langle p,g,x\rangle \mid p \textit{ is prime, } g \textit{ is a} \\ &\quad\ \textit{primitive root mod } p, \\ &\quad\ \textit{and } 1 \leq x \leq p-1\}, \end{aligned}$$

and for each $\langle p,g,x\rangle \in dom(h)$,

$$\begin{aligned} h(\langle p,g,x\rangle) &= \textit{the unique } c,\ 1 \leq c \leq p-1, \\ &\quad\ \textit{such that } g^c = x(\textit{mod } p). \end{aligned}$$

The computation has two components, (1) testing for membership in the domain and (2) the computation. Testing for membership in the domain is single-valued because the set of primes (and primitive roots) belongs to UP $\cap$ *co*UP *[FK92]. The discrete logarithm has a unique value when applied to tuples that belong to the domain.*

Given a partial multivalued function f, for all x, we define

$$set\text{-}f(x) = \{y \mid f(x) \mapsto y\}.$$

- FewPF is the set of all functions f in NPMV such that for some polynomial p and all x, $\|set\text{-}f(x)\| \leq p(|x|)$.

We take the point of view that a partial multivalued function is easy to compute if for each input string in the domain of the function, some value of the function is easy to compute. (We cannot compute all the values.) For this reason, we define the following technical notions. Given partial multivalued functions f and g, define g to be a *refinement* of f if $dom(g) = dom(f)$ and for all $x \in dom(g)$ and all y, if y is a value of $g(x)$, then y is a value of $f(x)$ (i.e., $set\text{-}g(x) \subseteq set\text{-}f(x)$). Let $\mathcal{F}$ and $\mathcal{G}$ be classes of partial multivalued functions. If f is a partial multivalued function, we define $f \in_c \mathcal{G}$ if $\mathcal{G}$ contains a refinement g of f, and we define $\mathcal{F} \subseteq_c \mathcal{G}$ if for every $f \in \mathcal{F}$, $f \in_c \mathcal{G}$. This notation is consistent with our intuition that $\mathcal{F} \subseteq_c \mathcal{G}$ should entail that the complexity of $\mathcal{F}$ is not greater than the complexity of $\mathcal{G}$. Thus, "NPMV $\subseteq_c$ PF" would mean that every partial multivalued function in NPMV can be computed efficiently by some deterministic polynomial time transducer. It is known [SXB83, Sel92, Sel94] that each of the following hypotheses are equivalent:

1. The function *sat* has a refinement in PF;
2. NPMV $\subseteq_c$ PF;
3. NPSV $\subseteq$ PF;
4. P = NP.

When $f \in$ FewPF, then it makes sense to seek all the values of $f(x)$. For a finite set $\{y_1, \cdots, y_n\}$, where the elements are listed in lexicographic order,

$$c(\{y_1, \cdots, y_n\}) = \%y_1 \cdots \%y_n\%,$$

where % is a symbol not in Σ. If $\|set\text{-}f(x)\|$ is finite for each x, then the function $c(set\text{-}f)$ is defined by $c(set\text{-}f)(x) = c(set\text{-}f(x))$. $c(set\text{-}f)$ is a single-valued total function. Given $f \in$ FewPF and a class of single-valued functions $\mathcal{G}$, define $f \in_c \mathcal{G}$ to mean that $c(set\text{-}f) \in \mathcal{G}$.

The class of partial functions that are computable in polynomial time with oracles in NP, PF^{NP}, has been well-studied [Kre88, Bei88], as have been the corresponding class of partial functions that can be computed nonadaptively with oracles in NP [Sel94], $\text{PF}^{\text{NP}}_{tt}$, and the classes of partial functions that are obtained by limiting the number of queries to some value $k \geq 1$, namely, $\text{PF}^{\text{NP}[k]}$ and $\text{PF}^{\text{NP}[k]}_{tt}$ [Bei91]. A rich body of results is known about these classes. (A partial function f is in $\text{PF}^{\text{NP}}_{tt}$ if there is an oracle

Turing machine transducer T such that $f \in \mathrm{PF}^{\mathrm{NP}}$ via T with an oracle L in NP and a polynomial time computable function $f: \{0,1\}^* \to (\%\{0,1\}^*)^*$ such that, for each input x to T, T only makes queries to L from the list $f(x)$.)

Let $\mathrm{PF}^{\mathrm{NP}}(O(\log\, n))$denote the class of functions computed in polynomial time with at most $O(\log\, n)$ queries to an oracle in NP. Krentel [Kre88] demonstrated that

$$\mathrm{PF}^{\mathrm{NP}} = \mathrm{PF}^{\mathrm{NP}}(O(\log\, n)) \text{ implies } \mathrm{P} = \mathrm{NP}.$$

Several of these classes seem to capture the complexity of computing NP-optimization problems [CT91, Kre88, War92, VW95, BKT94], but we will not explicity pursue this connection.

2.1 Inclusions

We know the following relations between these classes:

- $\mathrm{PF} \subseteq \mathrm{NPSV} \subseteq \mathrm{FewPF} \subseteq \mathrm{NPMV} \subseteq_c \mathrm{PF}^{\mathrm{NP}}$.
- $\mathrm{PF} \subseteq \mathrm{NPSV} \subseteq \mathrm{FewPF} \subseteq_c \mathrm{PF}^{\mathrm{NP}}_{tt} \subseteq \mathrm{PF}^{\mathrm{NP}}$.
- $\mathrm{PF} \subseteq \mathrm{PF}^{\mathrm{NP}}(O(\log\, n)) \subseteq \mathrm{PF}^{\mathrm{NP}}_{tt} \subseteq \mathrm{PF}^{\mathrm{NP}}$.

Most of these inclusions are obvious. The proof for language classes [Hem89, Wag90, BH91], shows that $\mathrm{PF}^{\mathrm{NP}}(O(\log\, n)) \subseteq \mathrm{PF}^{\mathrm{NP}}_{tt}$. To see that $\mathrm{FewPF} \subseteq_c \mathrm{PF}^{\mathrm{NP}}_{tt}$, we make the following definition. For each multivalued function f, define $code(f)$ to contain all tuples $\langle i, j, 0, x, k\rangle$, where $j \leq i$, such that there are at least i distinct values of f on x such that the j-th value in lexicographic order has a k-th bit, and to contain all tuples $\langle i, j, 1, x, k\rangle$, where $j \leq i$, such that there are at least i distinct values of f on x such that the k-th bit of the j-th value in lexicographic order is one. Then, for $f \in \mathrm{FewPF}$, it is easy to see that $code(f)$ belongs to NP and that all the values of f on input x can be computed nonadaptively in polynomial time from $code(f)$.

2.2 NP-search functions

Let $R(x,y)$ be an arbitrary relation in P (This is usually called an NP-relation.) and let p be a polynomial, so that the set

$$A = \{x \mid \exists y[|y| \leq p(|x|) \wedge R(x,y)]\}$$

belongs to NP. Define

$$f_{R,p}(x) \mapsto y, \text{ if } |y| \leq p(|x|) \wedge R(x,y).$$

The partial multivalued function $f_{R,p}$ is an "NP-search function."

Following Valiant [Val76], given a class of partial multivalued functions $\mathcal{F}$, let $\mathcal{F}_g$ denote the class of all $f \in \mathcal{F}$ such that $graph(f) \in \mathrm{P}$. Valiant noticed that ordinary search problems associated with NP decision problems are partial multivalued functions in NPMV_g. That is, the naturally occurring partial multivalued functions $f_{R,p}$ are in NPMV_g, and the function *sat* is a typical example. The converse is true as well. Every partial multivalued function in NPMV_g is the NP-search function of its graph.

Unless $\mathrm{P} = \mathrm{NP}$, not every function in NPMV belongs to NPMV_g. To see this, let $L \in \mathrm{NP} - \mathrm{P}$ and define the partial function S_L by $S_L(x) = 1$ if $x \in L$. ($S_L(x)$ is undefined for all $x \notin L$.) Observe that the partial function S_L belongs to NPSV and that $dom(S_L) = L$. It is easy to see that $graph(S_L) \in \mathrm{P}$ implies $L \in \mathrm{P}$. Thus, S_L does not belong to NPMV_g. The same argument proves that $\mathrm{NPSV} = \mathrm{NPSV}_g$ if and only if $\mathrm{P} = \mathrm{NP}$.

An interesting question is whether there are naturally occurring candidates for partial multivalued functions in NPMV that are not in NPMV_g, or in NPSV but not NPSV_g. The functions in Example 1 are in NPSV_g if primality testing is in P. Perhaps the reader knows whether these are likely candidates.

For any NP-search function $f_{R,p}$, let $maxf_{R,p}$ denote the partial function that on input x, outputs the lexicographically largest value of $f_{R,p}(x)$, if one exists. For every NP-relation R and polynomial p, $maxf_{R,p}$ belongs to the class $\mathrm{PF}^{\mathrm{NP}}$. The function *maxsat* is complete for $\mathrm{PF}^{\mathrm{NP}}$ [Kre88]. Precisely, given single-valued partial functions f and g, define $f \leq^{\mathrm{P}}_m g$ if there are partial functions $h_1, h_2 \in \mathrm{PF}$ such that for all $x \in dom(f)$, $f(x) = h_2(x, g(h_1(x)))$, and define $f \leq^{\mathrm{P}}_{tt} g$ if there are partial functions $h_1, h_2 \in \mathrm{PF}$ such that for all $x \in dom(f)$, $h_1(x) = \langle q_1, \ldots, q_k\rangle$, and $f(x) = h_2(x, \langle g(q_1), \ldots, g(q_k)\rangle)$. Krentel called the former *metric* reducibility and Watanabe and Toda [WT93] called the latter *functional* reducibility. In both cases, it is understood that $f(x)$ is defined only if every computation on the right side of the equation is defined. With these definitions in hand, *maxsat* is $\leq^{\mathrm{P}}_m$-complete for $\mathrm{PF}^{\mathrm{NP}}$. Watanabe and Toda raised the question of whether every single-valued refinement of *sat* is complete for $\mathrm{PF}^{\mathrm{NP}}$. They proved that if $\mathrm{NP} \neq co\mathrm{NP}$, then there is a single-valued refinement g of *sat* such that *maxsat* is not $\leq^{\mathrm{P}}_{tt}$-reducible to g. Thus, g is a single-valued refinement of *sat* that is not $\leq^{\mathrm{P}}_{tt}$-complete for $\mathrm{PF}^{\mathrm{NP}}$. We do not know whether their result holds for more general Turing reductions between partial functions.

Recall that a single-valued refinement of *sat* is a partial function f whose domain is the set of all satisfiable formulas such that for all $x \in \mathrm{SAT}$, $f(x)$ is a satisfying assignment. Much of the research on function classes has been motivated by the question of whether *sat* has a single-valued refinement in a smaller class than $\mathrm{PF}^{\mathrm{NP}}$ [WT93, HNOS94]. We will address this question as we proceed.

Observe that if A belongs to the class UP of problems that have unique solution [Val76], then for each x, there is at most one y such that $|y| \leq p(|x|) \wedge R(x,y)$. Thus, for A in UP, $f_{R,p}$ belongs to NPSV$_g$. Again, the converse is true; every partial function in NPSV$_g$ is the search function for a language in UP. Note however, by the argument above, that NPSV differs from NPSV$_g$ unless P = NP.

Jenner and Toran [JT96] provide an excellent survey of the subject from the point of view of examining the complexity of NP-search functions.

Given a class of partial multivalued functions $\mathcal{F}$, Valiant [Val76] defined $\mathcal{F}_t$ to denote the class of all $f \in \mathcal{F}$ that are defined on all input strings; i.e. $f \in \mathcal{F}_t$ if and only if $f \in \mathcal{F}$ and $dom(f) = \Sigma^*$. Beame *et al.* [BCE+95] study combinatorial properties of the class NPMV$_{tg}$. Let us say that an NP-acceptor N for SAT is *natural* if there is a function $f \in$ PF so that if $x \in$ SAT, and y is an accepting computation of N on input x, then $f(x,y)$ is a satisfying assignment of x. Fenner *et al.* [FFNR96] prove that NPMV$_t \subseteq_c$ PF if and only if every NP-acceptor for SAT is natural.

2.3 The complexity of inverting functions in PF

A function $f \in$ PF is *honest* if there is a polynomial q such that for every y in $range(f)$ there exists x in $dom(f)$ such that $f(x) = y$ and $|x| \leq q(|y|)$. The inverse of every honest function $f \in$ PF belongs to NPMV$_g$, and the inverse of every honest one-one function $f \in$ PF belongs to NPSV$_g$. (For any function f, the inverse f^{-1} is defined by $f^{-1}(x) \mapsto y$ if and only if $f(y) \mapsto x$.) The difficulty of inverting f is the complexity of the single-valued refinements of f^{-1}. A function f is *invertible in class $\mathcal{F}$*, where $\mathcal{F}$ is a class of functions, if f^{-1} has a single-valued refinement in $\mathcal{F}$. For example, f is invertible in polynomial time if f^{-1} has a single-valued refinement in PF. Every honest function in PF is invertible in PF$^{\mathrm{NP}}$. Every honest single-valued function f in PF is invertible in PF$_{tt}^{\mathrm{NP}}$. To see this, simply observe that for such a function f, f^{-1} is in NPSV$_g$, which is included in PF$_{tt}^{\mathrm{NP}}$.

We are interested in knowing whether every honest (one-one, few-one) function is invertible in some class that is smaller than PF$^{\mathrm{NP}}$. The following proposition [Sel94] addresses this question for several of the interesting cases. Let $\tau = \langle\ ,\ \rangle$ denote a polynomial time computable pairing function with polynomial time computable inverses σ_1 and σ_2.

Proposition 1 ([Sel94]) *Let $\mathcal{C}$ be any class of single-valued functions such that $f \in \mathcal{C}$ implies $\sigma_2(f) \in \mathcal{C}$. Then, every honest (one-one, few-one) polynomial time computable function is invertible in class $\mathcal{C}$ if and only if* NPMV$_g \subseteq \mathcal{C}$ *(*NPSV$_g \subseteq \mathcal{C}$, FewPF$_g \subseteq \mathcal{C}$, *respectively).*

All of the following are in part applications of this proposition.

Example 2 1. *Every honest polynomial time computable function is invertible in the class* PF

$$\begin{aligned} &\leftrightarrow \quad \mathrm{NPMV}_g \subseteq_c \mathrm{PF} \\ &\leftrightarrow \quad \mathrm{NPMV} \subseteq_c \mathrm{PF} \\ &\leftrightarrow \quad \mathrm{P} = \mathrm{NP}. \end{aligned}$$

2. *[GS88] Every honest one-one polynomial time computable function is invertible in the class* PF

$$\begin{aligned} &\leftrightarrow \quad \mathrm{NPSV}_g \subseteq_c \mathrm{PF} \\ &\leftrightarrow \quad \mathrm{P} = \mathrm{UP}. \end{aligned}$$

3. *Every honest few-one polynomial time computable function is invertible in the class* PF

$$\begin{aligned} &\leftrightarrow \quad \mathrm{FewPF}_g \subseteq_c \mathrm{PF} \\ &\leftrightarrow \quad \mathrm{P} = \mathrm{FewP}. \end{aligned}$$

4. *Every honest polynomial time computable function is invertible in the class* PF$_{tt}^{\mathrm{NP}}$

$$\begin{aligned} &\leftrightarrow \quad \mathrm{NPMV}_g \subseteq_c \mathrm{PF}_{tt}^{\mathrm{NP}} \\ &\leftrightarrow \quad \mathrm{NPMV} \subseteq_c \mathrm{PF}_{tt}^{\mathrm{NP}}. \end{aligned}$$

5. *Every honest polynomial time computable function is invertible in the class* NPSV

$$\begin{aligned} &\leftrightarrow \quad \mathrm{NPMV}_g \subseteq_c \mathrm{NPSV} \\ &\leftrightarrow \quad \mathrm{NPMV} \subseteq_c \mathrm{NPSV}. \end{aligned}$$

We will examine items 4 and 5 further as we proceed. For now, let us observe that these questions are especially interesting because NPMV $\subseteq_c$ PF$_{tt}^{\mathrm{NP}}$ if and only if every NP-search problem has a single-valued refinement in PF$_{tt}^{\mathrm{NP}}$ if and only if *sat* has a single-valued refinement in PF$_{tt}^{\mathrm{NP}}$, and NPMV $\subseteq_c$ NPSV if and only if every NP-search problem has a single-valued refinement in NPSV if and only if *sat* has a single-valued refinement in NPSV. This remark follows by noting that Cook's proof [Coo71] demonstrates that *sat* is complete for the functions in NPMV [Sel94].

2.4 Some interesting questions

2.4.1 PF$^{\mathrm{NP}} \subseteq$ PF$_{tt}^{\mathrm{NP}}$?

For any single-valued function f, we define $code(f)$ [Sel78] to be the set of all triples $\langle\sigma,x,k\rangle$, where $\sigma \in \{0,1\}$, such that the following properties hold: $\langle 0,x,k\rangle \in code(f)$

if and only if $f(x)$ has a k-th bit (i.e. $x \in dom(f)$ and $f(x)$ has length $\geq k$), and $\langle 1,x,k \rangle \in code(f)$ if and only if the k-th bit of $f(x)$ is 1. We say that a function f is *polynomial-bounded* if there is a polynomial p such that for all $x \in dom(f)$, $|f(x)| \leq p(|x|)$. For any single-valued polynomial-bounded function f, f can be nonadaptively computed in polynomial time using $code(f)$ as an oracle. The following proposition and theorem follow readily.

Proposition 2 ([Sel94]) **(i)** *$f \in \mathrm{PF}^{\mathrm{NP}}$ if and only if f is polynomial-bounded and* $code(f) \in \mathrm{P}^{\mathrm{NP}}$.

(ii) *$f \in \mathrm{PF}_{tt}^{\mathrm{NP}}$ if and only if f is polynomial-bounded and* $code(f) \in \mathrm{P}_{tt}^{\mathrm{NP}}$.

Theorem 1 ([Sel94]) $\mathrm{P}^{\mathrm{NP}} = \mathrm{P}_{tt}^{\mathrm{NP}}$ *if and only if* $\mathrm{PF}^{\mathrm{NP}} = \mathrm{PF}_{tt}^{\mathrm{NP}}$.

Beigel, Hemachandra, and Wechsung [BHW91] showed that $\mathrm{P}_{tt}^{\mathrm{NP}} \subseteq \mathrm{PP}$. Thus, $\mathrm{PF}^{\mathrm{NP}} = \mathrm{PF}_{tt}^{\mathrm{NP}}$ implies $\mathrm{P}^{\mathrm{NP}} \subseteq \mathrm{PP}$, which suggests that the classes $\mathrm{PF}^{\mathrm{NP}}$and $\mathrm{PF}_{tt}^{\mathrm{NP}}$are not identical. (There is an oracle relative to which P^{NP} is not a subset of PP [Bei94].)

Recall that $\mathrm{P}_{tt}^{\mathrm{NP}} = \mathrm{P}^{\mathrm{NP}}(O(\log n))$ [Hem89, Wag90, BH91]. Indeed, $\mathrm{P}^{\mathrm{NP}}(O(\log n))$ is a natural and robust complexity classes that has natural complete sets [Kre88, KSW87, Kad89, Wag90].

Since *maxsat* is complete for $\mathrm{PF}^{\mathrm{NP}}$, the question of whether $\mathrm{PF}^{\mathrm{NP}} \subseteq \mathrm{PF}_{tt}^{\mathrm{NP}}$ is equivalent to the question of whether *maxsat* belongs to the class $\mathrm{PF}_{tt}^{\mathrm{NP}}$. We learn from Proposition 2 that *maxsat* belongs to $\mathrm{PF}_{tt}^{\mathrm{NP}}$ if and only if for each satisfiable formula x, each bit of the maximum satisfying assignment can be computed independently and nonadaptively relative to NP. Then, from the result of Hemachandra, Wagner, and Buss and Hay, we learn that *maxsat* belongs to $\mathrm{PF}_{tt}^{\mathrm{NP}}$ if and only if each bit of the maximum satisfying assignment can be computed independently and without making more than $O(\log n)$ many queries to SAT.

2.4.2 NPMV $\subseteq_c \mathrm{PF}_{tt}^{\mathrm{NP}}$?

As we have seen already, whereas the previous question is equivalent to asking whether optimal solutions to NP-search problems can be computed nonadaptively using an oracle in NP, this question asks whether *any* single-valued refinement of NP-search problems can be computed in this manner. Also, recall (Example 2, item 4) that this question is equivalent to asking whether honest polynomial time computable functions are invertible in $\mathrm{PF}_{tt}^{\mathrm{NP}}$.

We turn first to a result of Buhrman, Kadin, and Thierauf [BKT94] that offers some insight into the question. Define the partial multivalued function *max_zero_sat* by *max_zero_sat*$(x) \mapsto y$ if and only if y is a satisfying assignment of x having the maximum number of 0's. Whereas the number of single-valued refinements of *sat* is large, these researchers prove that *sat* has a single-valued refinement in $\mathrm{PF}_{tt}^{\mathrm{NP}}$ if and only if *max_zero_sat* does. Furthermore, they prove that $\mathrm{PF}_{tt}^{\mathrm{NP}}$ is the class of all partial functions that are $\leq_m^{\mathrm{P}}$-reducible to some single-valued refinement of *max_zero_sat*. Of course, it is not known whether *max_zero_sat* is complete for $\mathrm{PF}_{tt}^{\mathrm{NP}}$, because it is not known whether any refinement of *sat* belongs to $\mathrm{PF}_{tt}^{\mathrm{NP}}$. It follows easily from these results that $\mathrm{PF}^{\mathrm{NP}} = \mathrm{PF}_{tt}^{\mathrm{NP}}$ if and only if *maxsat* is $\leq_m^{\mathrm{P}}$-reducible to some single-valued refinement of *max_zero_sat*.

Watanabe and Toda [WT93] prove that NPMV $\subseteq_c \mathrm{PF}_{tt}^{\mathrm{NP}}$ relative to a random oracle. In contrast, Burhman and Thierauf [BT96] have obtained the following results.

Theorem 2 ([BT96]) *If* E = NE *and sat* $\in_c \mathrm{PF}_{tt}^{\mathrm{NP}}$*, then every exponential-type search problem is solvable in* E*, where an "exponential-type search problem" denotes a search problem of a language in* NE.

Impagliazzo and Tardos [IT91] constructed an oracle A such that $\mathrm{E}^A = \mathrm{NE}^A$ but, relative to which, there exists an exponential-type search problem that cannot be solved in exponential time. Thus, as an immediate corollary, relative to this oracle A, $sat^A \notin_c (\mathrm{PF}^A)_{tt}^{\mathrm{NP}^A}$. So, there is an oracle relative to which NPMV does not have refinements in $\mathrm{PF}_{tt}^{\mathrm{NP}}$.

One might anticipate that the answer to this question is false, But as with most hypotheses that are known to relativize in both directions, the answer seems not to be forthcoming.

We have had better success with the question of whether NPMV $\subseteq_c$ NPSV (Example 2, item 5), for Hemaspaandra *et al.* [HNOS94] have shown that NPMV $\subseteq_c$ NPSV implies a collapse of the polynomial hierarchy. Recall that NPMV $\subseteq_c$ NPSV if and only if some single-valued refinement of *sat* belongs to NPSV, and if and only if every honest polynomial time computable function is invertible in NPSV. We will sketch the proof of this result and of subsequent recent developments in a later section. For now, the following comments are in order. The proof of Hemaspaandra *et al.* relativizes to all oracles. If the proof were to extend to show that NPMV $\subseteq_c \mathrm{PF}_{tt}^{\mathrm{NP}}$ implies a collapse of the polynomial hierarchy, and if the extension were to hold in all oracles, then, by the result of Watanabe and Toda that NPMV $\subseteq_c \mathrm{PF}_{tt}^{\mathrm{NP}}$ holds relative to a random oracle, it would follow that the polynomial hierarchy collapses relative to a random oracle. However, the polynomial hierarchy collapses relative to a random oracle only if the polynomial hierarchy collapses [Boo94]. Thus, such an extension of the proof of Hemaspaandra *et al.* would collapse the polynomial hierarchy as a corollary—an unlikely scenario.

2.4.3 $\mathrm{PF}_{tt}^{\mathrm{NP}} = \mathrm{PF}^{\mathrm{NP}}(O(\log\ n))$?

This is an especially intriguing question because much evidence indicates that that $\mathrm{PF}^{\mathrm{NP}}(O(\log\ n))$ is properly included in $\mathrm{PF}_{tt}^{\mathrm{NP}}$. Thus, since $\mathrm{P}_{tt}^{\mathrm{NP}} = \mathrm{P}^{\mathrm{NP}}(O(\log\ n))$, this question provides an excellent example for which relations between two function classes are different from for their corresponding language classes.

First we demonstrate that

$$\mathrm{PF}_{tt}^{\mathrm{NP}} = \mathrm{PF}^{\mathrm{NP}}(O(\log\ n)) \text{ implies } \mathrm{P} = \mathrm{FewP}.$$

Proposition 3 ([Sel94]) *The following statements are equivalent:*

1. $\mathrm{P} = \mathrm{FewP}$
2. $\mathrm{FewPF}_g \subseteq_c \mathrm{PF}$
3. $\mathrm{FewPF}_g \subseteq_c \mathrm{PF}^{\mathrm{NP}}(O(\log\ n))$

Proof. First we show that the first two statements are equivalent. The proof that $\mathrm{FewPF}_g \subseteq_c \mathrm{PF}$ implies $\mathrm{P} = \mathrm{FewP}$ is straightforward. Assume $\mathrm{FewP} = \mathrm{P}$. Let $f \in \mathrm{FewPF}_g$, let M be a nondeterministic transducer that computes f in time q, and let p bound $\|set\text{-}f\|$.

Consider the language

$$L = \{(x, c(F), u) \mid F \text{ is a finite set and there exists } w \notin F \text{ such that } u \text{ is a prefix of } w \text{ and } f(x) \mapsto w\}.$$

We claim that $L \in \mathrm{FewP}$: Given x, F, and u, guess a string w and check whether $w \notin F$, u is a prefix of w, and $(x,w) \in graph(f)$. The number of correct guesses is at most $p(|x|)$.

Since $\mathrm{FewP} = \mathrm{P}$ is assumed, $L \in \mathrm{P}$. For each x, the following algorithm uses L to compute $c(set\text{-}f(x))$ in polynomial time. The basic idea is to maintain F as a subset of $c(set\text{-}f(x))$. Use L to determine whether there exists a value of $f(x)$ that does not belong to L; if so, use L to find such a value w by implementing a typical prefix search, and then increment F to contain w.

begin
input x;
$F := \emptyset$;
while $(x, c(F), \lambda) \in L$ **do**
  begin { F is a proper subset of $set\text{-}f(x)$ }
  $u := \lambda$;
  while $((x,u) \notin graph(f) \vee u \in F)$ **do**
    if $(x, c(F), u0) \in L$
      then $u := u0$
      else if $(x, c(F), u1) \in L$
        then $u := u1$;
  { $f(x) \mapsto u \wedge u \notin F$ }
  $F := F \cup [u]$;
  end
halt in an accepting state with $c(F)$
on the output tape;
end.

When execution of the outer while-loop terminates, $F = set\text{-}f(x)$. To see this, note that the inner while-loop is reached only if there is a string $y \in set\text{-}f(x)$ that has not yet been found and that the inner while-loop preserves this property. In particular, the inner while-loop terminates only when a string u is found such that $f(x) \mapsto u$ and $u \notin F$. This condition ensures that if $f(x) \mapsto w_1$ and $f(x) \mapsto w_2$, where w_1 is a prefix of w_2, then both w_1 and w_2 are eventually placed into F.

Let us observe that the procedure runs in polynomial time. Since $L \in \mathrm{P}$ and $graph(f) \in \mathrm{P}$, each test takes polynomial time. The outer while-loop is executed at most $p(|x|)$ times, and, for each execution of the outer loop, the inner while-loop executes at most $q(|x|)$ times. Thus, we conclude that $c(set\text{-}f) \in \mathrm{PF}$.

Thus, the first two statements are equivalent. To see that statement 3 is equivalent to the other assertions, assume that $\mathrm{FewPF}_g \subseteq_c \mathrm{PF}^{\mathrm{NP}}(O(\log\ n))$ and let $f \in \mathrm{FewPF}_g$. Then, there is a $\mathrm{PF}^{\mathrm{SAT}}$ machine M that computes f and that makes at most $O(\log\ n)$ queries. Simulate M on input x for all possible oracle answers. This gives a polynomial number of possible output values. A value y belongs to $set\text{-}f(x)$ if and only if $(x,y) \in graph(f)$. Since $graph(f) \in \mathrm{P}$, $c(set\text{-}f) \in \mathrm{PF}$. □

Theorem 3 ([Sel94]) $\mathrm{PF}_{tt}^{\mathrm{NP}} = \mathrm{PF}^{\mathrm{NP}}(O(\log\ n))$ *implies* $\mathrm{P} = \mathrm{FewP}$.

The proof follows from Proposition 3. Recall that $\mathrm{FewPF} \subseteq_c \mathrm{PF}_{tt}^{\mathrm{NP}}$. Thus, the hypothesis implies that $\mathrm{FewPF}_g \subseteq \mathrm{FewPF} \subseteq_c \mathrm{PF}_{tt}^{\mathrm{NP}} \subseteq \mathrm{PF}^{\mathrm{NP}}(O(\log\ n))$, which, by the Proposition, implies $\mathrm{P} = \mathrm{FewP}$.

Next we demonstrate that

$$\mathrm{PF}_{tt}^{\mathrm{NP}} = \mathrm{PF}^{\mathrm{NP}}(O(\log\ n)) \text{ implies } \mathrm{R} = \mathrm{NP}.$$

The proof is an easy application of a result of Valiant and Vazirani [VV86]. Let SAT1 denote the set of formulas of propositional logic that have at most one satisfying assignment. Valiant and Vazirani showed that R = NP if the promise problem (SAT1, SAT) has a solution in P. By definition, a solution to the promise problem (SAT1, SAT) is any language L such that, for all x, if $x \in \mathrm{SAT1}$, then $x \in L \Leftrightarrow x \in \mathrm{SAT}$.

Theorem 4 ([Sel94]) $\mathrm{PF}_{tt}^{\mathrm{NP}} = \mathrm{PF}^{\mathrm{NP}}(O(\log n))$ *implies* $\mathrm{R} = \mathrm{NP}$.

Proof. Define

$$SAT' = \{(\phi, i) \mid \phi \text{ has } n \text{ variables}, n \geq i, \text{ and there is a satisfying assignment } w \text{ of } \phi \text{ in which the } i\text{-th variable is true}\}.$$

Clearly, $SAT' \in \mathrm{NP}$.

Define

$$cand(\phi) = SAT'((\phi,1))SAT'((\phi,2))\cdots SAT'((\phi,n)).$$

(One might think of *cand* as a candidate for a satisfying assignment of ϕ. Of course, in general it is unlikely.) Clearly, $cand \in \mathrm{PF}_{tt}^{\mathrm{NP}}$. Thus, by hypothesis, $cand \in \mathrm{PF}^{\mathrm{NP}}(O(\log n))$. Let M be a $\mathrm{PF}^{\mathrm{SAT}}$ machine that computes *cand* and that makes at most $O(\log n)$ queries.

Define M' to be a deterministic transducer that on an input ϕ simulates M for all possible oracle answers. As in the final part of the proof of Proposition 3, M''s output is a polynomial size list of values, and M' runs in polynomial time. Let

$$L = \{\phi \mid \text{some output value of } M' \text{ on input } \phi \text{ is a satisfying assignment}\}.$$

Then, $L \in \mathrm{P}$ and L is a solution of $(\mathrm{SAT1}, \mathrm{SAT})$. Thus, NP = R follows from the result of Valiant and Vazirani. □

This proof is similar to earlier applications of the result of Valiant and Vazirani by Beigel [Bei88] and Toda [Tod91].

Finally, we note that Jenner and Toran [JT95] proved that $\mathrm{PF}_{tt}^{\mathrm{NP}} = \mathrm{PF}^{\mathrm{NP}}(O(\log n))$ implies that for all $k > 0$, $\mathrm{SAT} \in \mathrm{DTIME}(2^{n/\log^k n})$. Their argument connects the hypothesis with a lowering of the amount of nondeterminism that is needed in a nondeterministic computation.

Although all of these results provide strong evidence that the two function classes $\mathrm{PF}_{tt}^{\mathrm{NP}}$ and $\mathrm{PF}^{\mathrm{NP}}(O(\log n))$ are not the same, it is not yet known whether $\mathrm{PF}_{tt}^{\mathrm{NP}} = \mathrm{PF}^{\mathrm{NP}}(O(\log n))$ implies $\mathrm{P} = \mathrm{NP}$. New work by Naik and Selman [NS96] reports modest progress on this question.

3 Reducibilities and Hierarchies

The purpose of this section is to define polynomial time-bounded Turing reducibility between partial multivalued functions and to explain the hierarchies that follow naturally. A hierarchy is conclusive demonstration that changes in computing resources impart changes in computing power. Thus, our philosophy is that a hierarchy is its own reward. That which here we merely sketch is developed fully by Fenner *et al.* [FHOS93].

Now we describe oracle Turing machines with oracles that compute partial functions. For the moment, we assume that the oracle g is a single-valued partial function. Let $\perp$ be a symbol not belonging to the finite alphabet Σ. In order for a machine M to access a partial function oracle, M contains a write-only input oracle tape, a separate read-only output oracle tape, and a special oracle call state q. When M enters state q, if the string currently on the oracle input tape belongs to the domain of the oracle partial function, then the result of applying the oracle appears on the oracle output tape, and if the string currently on the oracle input tape does not belong to the domain of the oracle partial function, then the symbol $\perp$ appears on the oracle output tape. Thus, given an input x to the oracle, the oracle, if called, returns a value of $g(x)$ if one exists, and returns $\perp$ otherwise. (It is possible that M may read only a portion of the oracle's output if the oracle's output is too long to read with the resources of M.) We shall assume, without loss of generality, that M never makes the same oracle query more than once, i.e., all of M's queries (on any possible computation path) are distinct.

If g is a single-valued partial function and M is a deterministic oracle transducer as just described, then we let $M[g]$ denote the single-valued partial function computed by M with oracle g.

Definition 1 *Let f and g be partial multivalued functions. We say that f* is polynomial-time Turing reducible to *g, denoted by $f \leq_{\mathrm{T}}^{\mathrm{P}} g$, if there is a deterministic oracle Turing machine M such that for every single-valued refinement g' of g, $M[g']$ is a single-valued refinement of f.*

The definition insists that the oracle g responds with a value of $g(x)$ whenever possible, which value does not matter; with this condition, M will compute some value of f on input x. This reducibility is reflexive and transitive over the class of all partial functions.

We can define nondeterministic reductions between partial functions with identical oracle access mechanism. In the case that g is a single-valued partial function and N is a polynomial time nondeterministic oracle Turing machine, then $N[g]$ denotes a partial multivalued function computed by N with oracle g in accordance with the prescription that when N asks about a value $y \in dom(g)$, then g returns $g(y)$, and when N asks about a value $y \notin dom(g)$, then g returns $\perp$. Observe that the function $f = N[g]$ is multivalued because N is nondeterministic. In the case that g is multivalued, the definition follows.

Definition 2 *Let f and g be partial multivalued functions. We say that f is* nondeterministic polynomial-time Turing reducible *to g, denoted by $f \leq_{\mathrm{T}}^{\mathrm{NP}} g$, if there is a polynomial time nondeterministic Turing machine N such that for every single-valued refinement g' of g, $N[g']$ is a refinement of f.*

Let $\mathcal{F}$ be a class of partial multivalued functions. $\mathrm{NPMV}^{\mathcal{F}}$ denotes the class of partial multivalued functions that are $\leq_{\mathrm{T}}^{\mathrm{NP}}$-reducible to some $g \in \mathcal{F}$.

For $k \geq 1$, define $\Sigma\mathrm{MV}_k$ inductively so that

$$\Sigma\mathrm{MV}_k = \underbrace{\mathrm{NPMV}^{\cdot^{\cdot^{\mathrm{NPMV}}}}}_{k}.$$

These classes form a function analogue of the polynomial hierarchy, and unless the polynomial hierarchy collapses, they form a proper hierarchy. Observe that a partial multivalued function f belongs to NPMV if and only if f is polynomial-bounded and $graph(f) \in \mathrm{NP}$. It is not hard to prove for $k \geq 1$, that $f \in \Sigma\mathrm{MV}_k$ if and only if f has a polynomial-bounded refinement g such that $graph(g) \in \Sigma_k^{\mathrm{P}}$. From this fact, the following theorem is straightforward.

Theorem 5 ([FHOS93]) *For every $k \geq 1$,*

$$\Sigma\mathrm{MV}_{k+1} = \Sigma\mathrm{MV}_k \Leftrightarrow \Sigma_{k+1}^{\mathrm{P}} = \Sigma_k^{\mathrm{P}}.$$

Now we return to consider the deterministic reduction. Let $\mathcal{F}$ be a class of partial multivalued functions. Then, $\mathrm{PF}^{\mathcal{F}}$ denotes the class of partial multivalued functions f that are $\leq_{\mathrm{T}}^{\mathrm{P}}$-reducible to some $g \in \mathcal{F}$. $\mathrm{PF}_{tt}^{\mathcal{F}}$ denotes the class of partial functions f that are $\leq_{\mathrm{T}}^{\mathrm{P}}$-reducible to some $g \in \mathcal{F}$ via an oracle Turing machine that queries its oracle *nonadaptively*. $\mathrm{PF}^{\mathcal{F}[k]}$ and $\mathrm{PF}_{tt}^{\mathcal{F}[k]}$ indicate that the number of queries is bounded by k, but we retain the notation $\mathrm{PF}^{\mathcal{F}}(O(\log\ n))$ for the class of functions that are computable with oracles in $\mathcal{F}$ by making $O(\log\ n)$ adaptive queries.

Identifying a language with its characteristic function, for any class of sets $\mathcal{C}$, the classes $\mathrm{PF}^{\mathcal{C}}$, $\mathrm{PF}_{tt}^{\mathcal{C}}$, and so on, are defined. Thus, we define such classes as $\mathrm{PF}^{\mathrm{NP}}$ and $\mathrm{PF}_{tt}^{\mathrm{NP}}$ as classes of partial multivalued functions. To see that $\mathrm{PF}^{\mathrm{NP}}$ contains partial functions that are not single-valued under this new definition, observe that $maxsat \in \mathrm{PF}^{\mathrm{NP}}$ and that $sat \leq_{\mathrm{T}}^{\mathrm{P}} maxsat$. Thus, $sat \in \mathrm{PF}^{\mathrm{NP}}$. Nevertheless, this extension and confusion of notation will not cause us problems. We will not go into the technical details here, but the inclusion relations that held before are exactly the ones that hold under the new interpretation of these class names.

Theorem 6 ([FHOS93]) $\mathrm{PF}^{\mathrm{NPMV}} = \mathrm{PF}^{\mathrm{NP}}$.

Proof. Obviously $\mathrm{PF}^{\mathrm{NP}} \subseteq \mathrm{PF}^{\mathrm{NPMV}}$. Conversely, for every function $f \in \mathrm{NPMV}$, $maxf$ is a single-valued refinement of f that belongs to $\mathrm{PF}^{\mathrm{NP}}$. So $\mathrm{NPMV} \subseteq \mathrm{PF}^{\mathrm{NPMV}}$. This implies that $\mathrm{PF}^{\mathrm{NPMV}} \subseteq \mathrm{PF}^{\mathrm{PF}^{\mathrm{NPMV}}} = \mathrm{PF}^{\mathrm{NPMV}}$ since $\leq_{\mathrm{T}}^{\mathrm{P}}$ is transitive. □

It is unlikely that $\mathrm{PF}_{tt}^{\mathrm{NPMV}}$ is the same as $\mathrm{PF}_{tt}^{\mathrm{NP}}$. Recall from Section 2.4.2 the centrality of the question of whether $\mathrm{PF}_{tt}^{\mathrm{NP}}$ contains a refinement of *max_zero_sat*. In contrast, *max_zero_sat* belongs to the class $\mathrm{PF}_{tt}^{\mathrm{NPMV}}$. Let f be a function that maps a pair $\langle x,n \rangle$ to y if and only if y is a satisfying assignment of x with n 0's. Since the number of variables in a formula is bounded by its length, it holds that $max\_zero\_sat(x) = f(x,n_x)$, where n_x is the largest n, $1 \leq n \leq |x|$, such that $(x,n) \in dom(f)$. This implies that $max\_zero\_sat \in \mathrm{PF}_{tt}^{\mathrm{NPMV}}$.

3.1 Bounded query hierarchy

The bounded adaptive and nonadaptive query hierarchies over NPMV are mostly analogous to those over NP. The reason seems to be that arguments that are reminiscent of the "mind-change" technique [Bei91, WW85] apply in this new setting. The basic results are the following:

Theorem 7 ([FHOS93]) *For every $k \geq 1$,*

$$\mathrm{PF}^{\mathrm{NPMV}[k]} = \mathrm{PF}_{tt}^{\mathrm{NPMV}[2^k-1]}.$$

We do not know whether $\mathrm{PF}^{\mathrm{NPMV}[k]} = \mathrm{PF}^{\mathrm{NP}[k]}$, but for reduction classes of sets, this is true.

Theorem 8 ([FHOS93]) *For every $k \geq 1$,*

$$\mathrm{P}^{\mathrm{NPMV}[k]} = \mathrm{P}^{\mathrm{NP}[k]}$$

and

$$\mathrm{P}_{tt}^{\mathrm{NPMV}[k]} = \mathrm{P}_{tt}^{\mathrm{NP}[k]}.$$

For every $k \geq 1$, $\mathrm{PF}^{\mathrm{NPMV}[k]} \subseteq \mathrm{PF}^{\mathrm{NPMV}[k+1]}$. The next results show that these classes form hierarchies.

Proposition 4 ([FHOS93]) *Let $k \geq 1$.*

1. *If $\mathrm{PF}^{\mathrm{NPMV}[k+1]} = \mathrm{PF}^{\mathrm{NPMV}[k]}$, then for every $\ell \geq k$, $\mathrm{PF}^{\mathrm{NPMV}[\ell]} = \mathrm{PF}^{\mathrm{NPMV}[k]}$.*
2. *If $\mathrm{PF}_{tt}^{\mathrm{NPMV}[k+1]} = \mathrm{PF}_{tt}^{\mathrm{NPMV}[k]}$, then for every $\ell \geq k$, $\mathrm{PF}_{tt}^{\mathrm{NPMV}[\ell]} = \mathrm{PF}_{tt}^{\mathrm{NPMV}[k]}$.*

The bounded adaptive and nonadaptive query hierarchies over NPMV collapse only if the Boolean hierarchy over NP collapses. The Boolean hierarchy over NP was defined by Wechsung and Wagner [WW85] and has been studied extensively [CGH+88, CGH+89, CH86, Kad88]. We denote the k-th level of the Boolean hierarchy as $\mathrm{NP}(k)$. Recall that

1. $\mathrm{NP}(1) = \mathrm{NP}$, and
2. for every $k \geq 2$, $\mathrm{NP}(k) = \mathrm{NP} - \mathrm{NP}(k-1)$.

The Boolean hierarchy over NP, denoted by BH is the union of all $\mathrm{NP}(k)$, $k \geq 1$. Kadin [Kad88] proved that the Boolean hierarchy collapses only if the polynomial-time hierarchy collapses.

Theorem 9 ([FHOS93]) *Let $k \geq 1$.*

1. *If $\mathrm{PF}_{tt}^{\mathrm{NPMV}[k+1]} = \mathrm{PF}_{tt}^{\mathrm{NPMV}[k]}$, then* BH *collapses to its $(k+1)$-st level.*
2. *If $\mathrm{PF}^{\mathrm{NPMV}[k+1]} = \mathrm{PF}^{\mathrm{NPMV}[k]}$, then* BH *collapses to its 2^k-th level.*

Proof. We prove the first statement. If

$$\mathrm{PF}_{tt}^{\mathrm{NPMV}[k+1]} = \mathrm{PF}_{tt}^{\mathrm{NPMV}[k]},$$

then by Proposition 4, for every $m > k$,

$$\mathrm{PF}_{tt}^{\mathrm{NPMV}[m]} \subseteq \mathrm{PF}_{tt}^{\mathrm{NPMV}[k]}.$$

So, by Theorem 8, the query hierarchy over NP collapses: for every $m > k$, $\mathrm{P}_{tt}^{\mathrm{NP}[m]} = \mathrm{P}_{tt}^{\mathrm{NP}[k]}$. Thus by results of Köbler, Schöning, and Wagner [KSW87] that show that levels of the nonadaptive query hierarchy over NP and levels of the Boolean hierarchy interleave, for every $m > k$,

$$\mathrm{P}_{tt}^{\mathrm{NP}[m]} = \mathrm{P}_{tt}^{\mathrm{NP}[k]} \subseteq \mathrm{NP}(k+1).$$

Thus, $\mathrm{BH} = \mathrm{NP}(k+1)$. □

Finally, because the mind-change argument works using NPMV as the oracle class, we have the following result. This result stands in strong contrast to the results of Section 2.4.3.

Theorem 10 ([FHOS93])

$$\mathrm{PF}^{\mathrm{NPMV}[\log]} = \mathrm{PF}_{tt}^{\mathrm{NPMV}}.$$

3.2 Single-valued oracles

Most of the results that we stated in the previous section for reductions to NPMV hold as well if the oracle is NPSV. However, comparison of the adaptive and nonadaptive query hierarchies is problematical. The following summarize the known results.

Theorem 11 ([FHOS93]) *The following statements hold.*

1. $\mathrm{PF}^{\mathrm{NPSV}} = \mathrm{PF}^{\mathrm{NP}}$.
2. $\mathrm{P}^{\mathrm{NPSV}} = \mathrm{P}^{\mathrm{NP}}$.
3. $\mathrm{P}_{tt}^{\mathrm{NPSV}} = \mathrm{P}_{tt}^{\mathrm{NP}}$.
4. *For all $k \geq 1$, $\mathrm{P}^{\mathrm{NPSV}[k]} = \mathrm{P}^{\mathrm{NP}[k]}$ and*
$$\mathrm{P}^{\mathrm{NPSV}}(O(\log n)) = \mathrm{P}^{\mathrm{NP}}(O(\log n)).$$
5. *For all $k \geq 1$, $\mathrm{P}_{tt}^{\mathrm{NPSV}[k]} = \mathrm{P}_{tt}^{\mathrm{NP}[k]}$.*

Both the adaptive and nonadaptive NPSV query hierarchies are proper unless the Boolean and polynomial hierarchies collapse.

Theorem 12 ([FHOS93]) *Let $k \geq 1$.*

1. *If $\mathrm{PF}_{tt}^{\mathrm{NPSV}[k+1]} = \mathrm{PF}_{tt}^{\mathrm{NPSV}[k]}$, then* BH *collapses to its $(k+1)$-st level.*
2. *If $\mathrm{PF}^{\mathrm{NPSV}[k+1]} = \mathrm{PF}^{\mathrm{NPSV}[k]}$, then* BH *collapses to its 2^k-th level.*

As a consequence, because *maxsat* is complete for $\mathrm{PF}^{\mathrm{NPMV}}$, for any $k \geq 1$, if *maxsat* $\in \mathrm{PF}^{\mathrm{NPSV}[k]}$, then the Boolean and polynomial hierarchies collapse. In the next section we will learn that the polynomial hierarchy collapses if for any $k \geq 1$, *any* refinement of *sat* belongs to $\mathrm{PF}^{\mathrm{NPSV}[k]}$ [Ogi95].

Comparing adaptive with nonadaptive classes, we know only the following:

Theorem 13 ([FHOS93]) *The following inclusions hold.*

1. $\mathrm{PF}^{\mathrm{NPSV}[k]} \subseteq \mathrm{PF}_{tt}^{\mathrm{NPSV}[2^k-1]}$.
2. $\mathrm{PF}^{\mathrm{NPSV}}(O(\log n)) \subseteq \mathrm{PF}_{tt}^{\mathrm{NPSV}}$.

It is an open question whether

$$\mathrm{PF}^{\mathrm{NPSV}}(O(\log n)) = \mathrm{PF}_{tt}^{\mathrm{NPSV}}.$$

In this regard, is NPSV more like NPMV or more like NP? Whereas $\mathrm{PF}_{tt}^{\mathrm{NPSV}} = \mathrm{PF}_{tt}^{\mathrm{NP}}$, apparently, $\mathrm{PF}^{\mathrm{NPSV}}(O(\log n))$ and $\mathrm{PF}^{\mathrm{NP}}(O(\log n))$ are not equal.

$$\mathrm{NPSV} \subseteq \mathrm{PF}^{\mathrm{NPSV}[1]} \subseteq \mathrm{PF}^{\mathrm{NPSV}}(O(\log n)),$$

and $\mathrm{NPSV} \subseteq \mathrm{PF}^{\mathrm{NP}}(O(\log n))$ implies $\mathrm{P} = \mathrm{UP}$. The implication follows from the fact that $\mathrm{P} = \mathrm{UP}$ if and only if $\mathrm{NPSV}_g = \mathrm{PF}^{\mathrm{NP}}(O(\log n))$, the proof of which is similar to the proof of Proposition 3. Thus,

$$\mathrm{PF}^{\mathrm{NPSV}}(O(\log n)) \subseteq \mathrm{PF}^{\mathrm{NP}}(O(\log n)) \Rightarrow \mathrm{P} = \mathrm{UP}.$$

Similarly,

$$\mathrm{PF}^{\mathrm{NPMV}[1]} \subseteq \mathrm{PF}^{\mathrm{NP}}(O(\log n)) \Rightarrow \mathrm{P} = \mathrm{NP}.$$

3.3 Difference hierarchy

In analogy to the Boolean hierarchy of languages, we define a *difference hierarcy* of partial multivalued functions. This hierarchy is defined by Fenner *et al.* [FHOS93] and further developed in the new paper of Fenner *et al.* [FGH$^+$96]. The difference hierarchy is defined so that for each $k \geq 1$, $f \in \mathrm{NPMV}(k)$ if and only if f is polynomial-bounded and $graph(f) \in \mathrm{NP}(k)$. As a consequence, for every $k \geq 1$, $\mathrm{NPMV}(k+1) = \mathrm{NPMV}(k)$ if and only if $\mathrm{NP}(k+1) = \mathrm{NP}(k)$. However the contour of the difference hierarchy over NPMV is astonishingly different from the Boolean hierarchy over NP. For example, whereas the levels of the Boolean hierarchy interleave with those of the bounded query hierarchy over NP, and sit properly within $\mathrm{PF}^{\mathrm{NP}}$, the function *maxsat*, which recall is complete for $\mathrm{PF}^{\mathrm{NP}}$, belongs to NPMV(2). We leave it to the paper of Fenner *et al.* [FGH$^+$96] to explain the reason for this.

4 Number of output values

Define the NPkV hierarchy as follows. For all $k \geq 1$, a partial multivalued function $f \in \mathrm{NP}k\mathrm{V}$ if some refinement of f can be computed by a polynomial time-bounded transducer that has at most k distinct values on any input. Thus, in particular, NP1V = NPSV. In his Ph.D. dissertation, Naik [Nai94] raised the question of whether the NPkV hierarchy is proper, that is, whether for all $k \geq 1$, $\mathrm{NP}(k+1)\mathrm{V} \neq \mathrm{NP}k\mathrm{V}$. As supporting evidence of that possibility he proved that this hierarchy is proper relative to a random oracle.

Hemaspaandra *et al.* [HNOS94] effectively settled the question of whether every function in NPMV has a single-valued refinement in NPSV (Recall the discussion in Section 2.4.2.) by showing that this hypothesis collapses the polynomial hierarchy. Precisely, Hemaspaandra *et al.* proved the following theorem.

Theorem 14 ([HNOS94]) *If* $\mathrm{NP2V} \subseteq_c \mathrm{NPSV}$, *then* $\mathrm{PH} = \Sigma_2^{\mathrm{P}}$.

Recently Ogihara [Ogi95] has extended the result of Hemaspaandra *et al.* to show the following.

Theorem 15 ([Ogi95]) *Let $c < 1$ be a constant. If every multivalued function in* NPMV *has a refinement in* $\mathrm{PF}^{\mathrm{NPSV}}(c\log\ n)$, *then* $\mathrm{PH} = \Sigma_2^{\mathrm{P}}$.

$\mathrm{NPMV} \subseteq_c \mathrm{PF}^{\mathrm{NPSV}}(c\log\ n)$ if and only if $sat \in_c \mathrm{PF}^{\mathrm{NPSV}}(c\log\ n)$. As a consequence, if for some constant $k \geq 1$, *sat* has a refinement in $\mathrm{PF}^{\mathrm{NPSV}[k]}$, then the polynomial hierarchy collapses.

Naik [Nai95] has observed that Ogihara's proof can be modified to show that Theorem 14 holds for all $k \geq 1$. That is, the following theorem holds.

Theorem 16 ([Nai95]) *Let $k > 1$. If*

$$\mathrm{NP}k\mathrm{V} \subseteq_c \mathrm{NP}(k-1)\mathrm{V},$$

then $\mathrm{PH} = \Sigma_2^{\mathrm{P}}$.

Now we will describe the proof of Theorem 16, Naik's version of Ogihara's theorem. To begin, we want a partial multivalued function f that obviously belongs to the class NPkV but that intuitively has no refinement in $\mathrm{NP}(k-1)\mathrm{V}$. This consideration leads us into the world of selectivity. Let us say that a set A is *k-selective* if there is a partial multivalued function f (We call f a k-selector of A.) such that

1. input to f is a set $Y \subseteq \Sigma^*$ such that $\|Y\| = k$,
2. every output value of $f(Y)$ is a set Z such that $Z \subseteq Y$ and $\|Z\| = k-1$, and
3. if at least $k-1$ of the strings in Y belong to A, then $set\text{-}f(Y) \neq \emptyset$ and
$$Z \in set\text{-}f(Y) \Rightarrow Z \subseteq A.$$

Example 3 *Let k = 2. Then A is 2-selective if there is a partial multivalued function f defined on ordered pairs such that*

$$set\text{-}f(x,y) \subseteq \{x,y\}$$

and such that if $x \in A$ or $y \in A$, then

$$set\text{-}f(x,y) \neq \emptyset \text{ and } set\text{-}f(x,y) \subseteq A.$$

For those familiar with previous discources on selectivity, we note that a set A is p-selective [Sel79] if A has a 2-selector that belongs to PF_t and that A is NPMV-selective if A has a 2-selector that belongs to NPMV [HNOS94].

Let $k > 1$. We claim that every set $A \in \mathrm{NP}$ has a k-selector that belongs to NPkV. Define f on input Y, where $\|Y\| = k$ so that f nondeterministically guesses a subset Z of $k-1$ strings. Then, f nondeterministically tries to discover whether $Z \subseteq A$. If this test is successful, then f outputs the set Z. Since Y has k distinct subsets of size $k-1$, we see that $f \in \mathrm{NP}k\mathrm{V}$.

The reader can easily see that if f is a k-selector for A and g is a refinement of f, then g is a k-selector for A. Assume as hypothesis that $\mathrm{NP}k\mathrm{V} \subseteq_c \mathrm{NP}(k-1)\mathrm{V}$. Then, A has a k-selector g that belongs to $\mathrm{NP}(k-1)\mathrm{V}$. We will infer from this assumption that $\Pi_2^{\mathrm{P}} = \Sigma_2^{\mathrm{P}}$.

Example 3, continued *Let A be* SAT *and let f be a 2-selector for* SAT *that belongs to* NPMV. *A single-valued refinement of f is a single-valued partial function g such that if either $x \in$* SAT *or $y \in$* SAT, *then $g(x,y)$ is defined and $g(x,y) \in$* SAT.

Intuitively, one does not expect a single-valued function to be able to determine which of two formulas is satisfiable. This intuition is borne out by the result of Selman [Sel79] that SAT is p-selective if and only if SAT $\in$ P and by the result of Hemaspaandra et al. [HNOS94] that SAT is NPSV-selective only if NP $\subseteq$ (NP $\cap$ *co*NP)/*poly*.

Continuing with the proof, let $L \in \Pi_2^{\mathrm{P}}$. There exists a polynomial p and a set $A \in$ NP such that

$$x \in L \Leftrightarrow \forall y \in \Sigma^{p(|x|)}, \langle x,y \rangle \in A.$$

We may assume that there is a polynomial q such that for all strings x of length n and all strings y of length $p(n)$, $|\langle x,y\rangle| = q(n)$. Let $A^{=q(n)} = A \cap \Sigma^{=q(n)}$. We are assuming that A has a k-selector g that belongs to NP$(k-1)$V. Given a string $x \in \Sigma^{=q(n)}$ and a set $Z \subseteq A^{=q(n)}$ such that $\|Z\| = k-1$, we say that x *loses to* Z if every output value of $g(Z \cup \{x\})$ contains x.

If x loses to Z, then $x \in A$: Since $Z \subseteq A^{=q(n)}$, *set*-$g(Z \cup \{x\}) \neq \emptyset$. Furthermore, for every output value Y, $Y \subseteq A$. Thus, for each such Y, $x \in Y \subseteq A$.

Example 3, continued *For $z \in \mathrm{SAT}^{=n}$, a string x loses to z if $g(x,z) = x$. By definition of a selector, $x \in \mathrm{SAT}$ follows.*

Lemma 1 *For every $n \geq 1$, there is a set $S_{q(n)} = \{Z_1, \ldots, Z_m\}$, $m \leq q(n)$, such that for every i, $1 \leq i \leq m$, $Z_i \subseteq A^{=q(n)}$, $\|Z_i\| = k-1$, and for all $x \in \Sigma^{q(n)}$, $x \in A^{=q(n)}$ if and only if there exists i, $1 \leq i \leq m$, such that x loses to Z_i.*

Example 3, concluded *Lemma 1 asserts the existence of a set of strings $S_{q(n)} = \{z_1, \ldots, z_m\}$, $m \leq q(n)$, such that $S_{q(n)} \subseteq \mathrm{SAT}^{=q(n)}$, and for all $x \in \mathrm{SAT}^{=q(n)}$, there exists i, $1 \leq i \leq m$, such that $g(x,z) = x$.*

We will not give the proof of Lemma 1. The proof is similar to the proof of Ko [Ko83] and of later researchers [LS93, HNOS94] that dealt essentially with the scenario of Example 3. The combinatorics of Ogihara's argument is necessarily more involved. The key idea of the proof is to note that some set Z is a winner to more than the average number of strings x (meaning that x loses to Z). Put such a Z into $S_{q(n)}$, delete from consideration all strings that lose to Z, and continue the process.

Define a string u to be *correct for length $q(n)$* if u encodes a tuple $\langle S, WIT\rangle$ such that $S = \{Z_1, \ldots, Z_m\}$ and $WIT = \{W_1, \ldots, W_m\}$, $m \leq q(n)$, that satisfy the following conditions:

(i) for all i, $1 \leq i \leq m$, $\|Z_i\| = k-1$,

(ii) for all i, $1 \leq i \leq m$, $Z_i \subseteq A^{=q(n)}$ and W_i is a set of witnesses proving that $Z_i \subseteq A^{=q(n)}$, and

(iii) for all $x \in A^{=q(n)}$ there exists i, $1 \leq i \leq m$, such that x loses to Z_i.

If u is correct for length $q(n)$, we write $\mathrm{Loses}(x,u)$ to mean x loses to some Z_i.

Then,

$$\begin{aligned} x \in L \quad \Leftrightarrow \quad & \exists u[(u \text{ is correct for } q(|x|)) \text{ and} \\ & \forall y\, \mathrm{Loses}(\langle x,y\rangle, u)]. \end{aligned}$$

The implication from left to right follows from Lemma 1. The implication from right to left is straightforward.

To complete the argument that $L \in \Sigma_2^{\mathrm{P}}$, we merely have to prove that the predicates

1. "u is correct for $q(|x|)$," and
2. $\mathrm{Loses}(\langle x,y\rangle, u)$

are in *co*NP.

To prove that "u is correct for $q(|x|)$" belongs to *co*NP, we give an NP-algorithm for the complement "u is not correct for $q(|x|)$." If u does not encode a tuple $\langle S, WIT\rangle$ that satisfies the defining conditions (i) and (ii), then accept. Otherwise, $S = \{Z_1, \ldots, Z_m\}$ and for each i, $Z_i \subseteq A^{=q(n)}$. Thus, and this is the important observation, for each $x \in \Sigma^{q(n)}$ and each Z_i, $g(Z_i \cup \{x\})$ is defined. Nondeterministically select $x \in A^{=q(n)}$. For each i, compute an output value Y of $g(Z_i \cup \{x\})$ and verify that $x \notin Y$. If each of these tests is successful, then accept.

The proof that the second predicate belongs to *co*NP is similar.

This completes the proof of Theorem 16.

5 Open problems

1. Let $k \geq 1$. Does NP$(k+1)$V $\subseteq_c$ NPkV imply for all $m \geq k$, NPmV $\subseteq_c$ NPkV?
2. Clearly, NP $=$ *co*NP implies NPMV $\subseteq$ NPSV. But, in general, does a collapse of the polynomial hierarchy imply a collapse of the NPkV hierarchy? If so, then, since the NPkV hierarchy is infinite relative to a random oracle, it would follow that the polynomial hierarchy is infinite relative to a random oracle.
3. Define UP_k to be the class of all languages in NP that are acceptable by an NP-machine that has at most k accepting computations on every input. One can associate each language $L \in \mathrm{UP}_k$ with the partial function in NPkV$_g$ that maps each $x \in L$ to the accepting computations of the UP_k-acceptor for L. For $k \geq 1$, does $\mathrm{UP}_{k+1} = \mathrm{UP}_k$ imply that the polynomial hierarchy collapses? Does UP $=$ NP imply that the polynomial hierarchy collapses? The results about function

classes seem not to imply anything about the corresponding language classes. The problem is that some strange unambiguous Turing machine might accept SAT whose accepting paths have no connection with the problem of computing satisfying assignments. If, however, every machine that accepts SAT is natural (as defined in Section 2.2), then UP = NP implies that NPMV $\subseteq_c$ NPSV. Hence, under the hypothesis that every machine that accepts SAT is natural, UP = NP implies that the polynomial hierarchy collapses.

In this regard we mention that the proof technique that yields a random oracle relative to which the NPkV hierarchy is infinite also demonstrates that UP $\neq$ NP relative to a random oracle, but does not seem to suffice to separate the classes UP_{k+1} and UP_k.

4. Another related open question is whether a conjecture raised by Even, Selman, and Yacobi [ESY84] holds relative to a random oracle. The conjecture states that for all disjoint Turing-complete sets A and B in NP, there exists a set C such that C separates A and B and C is *not* Turing-hard for NP. It is known [ESY84, GS88, Sel94] that this conjecture implies (i) NP $\neq$ *co*NP, (ii) NP $\neq$ UP, and (iii) NPMV $\not\subseteq_c$ NPSV. Each of the these consequences holds relative to a random oracle [BG81, Roy94, Nai94]. In fact, relative to a random oracle, the same language separates the classes in (i) and (ii), and a search function of this language separates (iii).

References

[ÀJ93] C. Àlvarez and B. Jenner. A very hard log-space counting class. *Theoret. Comput. Sci.*, 107:3–30, 1993.

[BCE+95] P. Beame, S. Cook, J. Edmonds, R. Impagliazzo, and T. Pitassi. The relative complexity of NP search problems. In *Proc. 27th ACM Symp. on Theory of Computing*, pages 303–314, 1995.

[Bei88] R. Beigel. NP-hard sets are P-superterse unless R = NP. Technical Report 88-04, Department of Computer Science, The Johns Hopkins University, 1988.

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

[Bei94] R. Beigel. Perceptrons, PP, and the polynomial hierarchy. *Computational Complexity*, 4:339–349, 1994.

[BG81] C. Bennett and J. Gill. Relative to a random oracle A, $\mathrm{P}^A \neq \mathrm{NP}^A \neq \text{Co-NP}^A$ with probability 1. *SIAM J. Comput.*, 10(1):96–113, February 1981.

[BH91] S. Buss and L. Hay. On truth table reducibility to SAT. *Information and Computation*, 91(1):86–102, 1991.

[BHW91] R. Beigel, L. Hemachandra, and G. Wechsung. Probabilistic polynomial time is closed under parity reductions. *Information Processing Letters*, 37(2):91–94, 1991.

[BKT94] H. Buhrman, J. Kadin, and T. Thierauf. On functions computable with nonadaptive queries to NP. In *Proc. 9th IEEE Conference on Structure in Complexity Theory*, pages 43–52, 1994.

[BLS84] R. Book, T. Long, and A. Selman. Quantitative relativizations of complexity classes. *SIAM J. Comput.*, 13(3):461–487, August 1984.

[Boo94] R. Book. On collapsing the polynomial-time hierarchy. *Information Processing Letters*, 52:235–237, 1994.

[BT96] H. Buhrman and T. Thierauf. The complexity of generating and checking proofs of membership. In *Proc. 13th Symp. on Theoretical Aspects of Computer Science, Lecture Notes in Computer Science*. Springer-Verlag, 1996.

[CGH+88] J. Cai, T. Gundermann, J. Hartmanis, L. Hemachandra, V. Sewelson, K. Wagner, and G. Wechsung. The boolean hierarchy I: Structural properties. *SIAM J. Comput.*, 17(6):1232–1252, 1988.

[CGH+89] J. Cai, T. Gundermann, J. Hartmanis, L. Hemachandra, V. Sewelson, K. Wagner, and G. Wechsung. The boolean hierarchy II: Applications. *SIAM J. Comput.*, 18(1):95–111, 1989.

[CH86] J. Cai and L. Hemachandra. The Boolean hierarchy: Hardware over NP. In *Structure in Complexity Theory, Lecture Notes in Computer Science 223*, pages 105–124, Berlin, 1986. Springer-Verlag.

[CLL+95] J. Cai, R. Lipton, L. Longpré, M. Ogihara, K. Regan, and D. Sivakumar. Communication complexity of key agreement on limited ranges. In *Proc. 12th Symp. on Theoretical Aspects of Computer Science, Lecture Notes in Computer Science*, pages 38–49. Springer-Verlag, 1995.

[Coo71] S. Cook. The complexity of theorem-proving procedures. In *Proc. 3rd ACM Symp. on Theory of Computing*, pages 151–158, 1971.

[CT91] Z. Chen and S. Toda. On the complexity of computing optimal solutions. *International J. of Foundations of Computer Science*, 2:207–220, 1991.

[ESY84] S. Even, A. Selman, and Y. Yacobi. The complexity of promise problems with applications to public-key cryptography. *Information and Control*, 61(2):159–173, May 1984.

[FFNR96] S. Fenner, L. Fortnow, A. Naik, and J. Rogers. On inverting onto functions. In *Proc. 11th IEEE Conference on Computational Complexity*, 1996.

[FGH$^+$96] S. Fenner, F. Green, S. Homer, A. Selman, T. Thierauf, and H. Vollmer. Complements of multivalued functions. In *Proc. 11th IEEE Conference on Computational Complexity*, 1996.

[FHOS93] S. Fenner, S. Homer, M. Ogihara, and A. Selman. On using oracles that compute values. In *Proc. 10th Symp. on Theoretical Aspects of Computer Science, Lecture Notes in Computer Science*, volume 665, pages 398–407. Springer-Verlag, 1993.

[FK92] M. Fellows and N. Koblitz. Self-witnessing polynomial-time complexity and prime factorization. In *Proc. 7th IEEE Conference on Structure in Complexity Theory*, pages 107–110, 1992.

[GS88] J. Grollmann and A. Selman. Complexity measures for public-key cryptosystems. *SIAM J. Comput.*, 17(2):309–355, April 1988.

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

[HNOS94] L. Hemaspaandra, A. Naik, M. Ogihara, and A. Selman. Computing unique solutions collapses the polynomial hierarchy. In D. Du and X. Zhang, editors, *Proc. 5th International Symp. on Algorithms and Computation, Lecture Notes in Computer Science*, pages 56–64. Springer-Verlag, 1994.

[IT91] R. Impagliazzo and G. Tardos. Search versus decision in super-polynomial time. In *Proc. 32nd IEEE Foundations of Computer Science*, pages 222–227, 1991.

[JT95] B. Jenner and J. Torán. Computing functions with parallel queries to NP. *Theoret. Comput. Sci.*, 141, 1995.

[JT96] B. Jenner and J. Torán. The complexity of obtaining solutions for problems in NP. In L. Hemaspaandra and A. Selman, editors, *Complexity Theory Retrospective II*. Springer-Verlag, New York, 1996.

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

[Kad89] J. Kadin. $\mathrm{P}^{\mathrm{NP}}[\log\ n]$ and sparse Turing-complete sets for NP. *J. of Computer and System Sciences*, 39(3):282–298, 1989.

[Ko83] K. Ko. On self-reducibility and weak P- selectivity. *J. of Computer and System Sciences*, 26:209–211, 1983.

[Kre88] M. Krentel. The complexity of optimization problems. *J. of Computer and System Sciences*, 36:490–509, 1988.

[KSW87] J. Köbler, U. Schöning, and K. Wagner. The difference and truth-table hierarchies for NP. *Theoretical Informatics and Applications (RAIRO)*, 21:419–435, 1987.

[LS93] L. Longpré and A. Selman. Hard promise problems and nonuniform complexity. *Theor. Comput. Sci.*, 115(3):277–290, 1993.

[Moc93] S. Mocas. *Separating exponential time classes from polynomial time classes*. PhD thesis, Northeastern University, Boston, MA, 1993.

[Nai94] A. Naik. *The structural complexity of intractable search functions*. PhD thesis, State University of New York at Buffalo, Buffalo, NY, 1994.

[Nai95] A. Naik. Personal communication. 1995.

[NS96] A. Naik and A. Selman. A note on p-selective sets and on adaptive versus nonadaptive queries to np. In *Proc. 11th IEEE Conference on Computational Complexity*. 1996.

[Ogi95] M. Ogihara. Functions computable with multiple access to NP. Technical Report TR-583, Department of Computer Science, University of Rochester, Rochester, NY, 1995.

[OR93] M. Ogihara and K. Regan. Random polynomial time computable functions. manuscript, 1993.

[Roy94] J. Royer. Personal communication. 1994.

[Sel78] A. Selman. Polynomial time enumeration reducibility. *SIAM J. Comput.*, 7(4):440–457, November 1978.

[Sel79] A. Selman. P-selective sets, tally languages, and the behavior of polynomial time reducibilities on NP. *Math. Systems Theory*, 13:55–65, 1979.

[Sel92] A. Selman. A survey of one-way functions in complexity theory. *Math. Systems Theory*, 25:203–221, 1992.

[Sel94] A. Selman. A taxonomy of complexity classes of functions. *J. of Computer and System Sciences*, 48(2):357–381, 1994.

[SXB83] A. Selman, Xu M.-R., and R. Book. Positive relativizations of complexity classes. *SIAM J. Comput.*, 12:465–479, 1983.

[Tod91] S. Toda. On polynomial-time truth-table reducibilities of intractable sets to P-selective sets. *Math. Systems Theory*, 24(2):69–82, 1991.

[Val76] L. Valiant. Relative complexity of checking and evaluating. *Information Processing Letters*, 5(1):20–23, May 1976.

[VV86] L. Valiant and V. Vazirani. NP is as easy as detecting unique solutions. *Theoret. Comput. Sci.*, 47:85–93, 1986.

[VW95] H. Vollmer and K. Wagner. Complexity classes of optimization functions. *Information and Computation*, 120:198–219, 1995.

[Wag90] K. Wagner. Bounded query classes. *SIAM J. Comput.*, 19:833–846, 1990.

[War92] H. Wareham. On the comptutational complexity of inferring evolutionary trees. Master's thesis, Department of Computer Science, Memorial University of Newfoundland, 1992.

[WT93] O. Watanabe and S. Toda. Structural analysis of the complexity of inverse functions. *Math. Systems Theory*, 26:203–214, 1993.

[WW85] G. Wechsung and K. Wagner. On the boolean closure of NP. In *Proc. International Conf. on Fundamentals of Computation Theory, Lecture Notes in Computer Science 199*, pages 485–493. Springer-Verlag, Berlin, 1985.