SELF-REDUCIBILITY: THE EFFECTS OF STRUCTURE ON COMPLEXITY

by

Deborah Joseph
Paul Young

Computer Sciences Technical Report #798

October 1988

STRUCTURAL COMPLEXITY GUEST COLUMN
Bulletin of the EATCS

SELF-REDUCIBILITY: THE EFFECTS OF STRUCTURE ON COMPLEXITY

DEBORAH JOSEPH
University of Wisconsin - Madison

PAUL YOUNG
University of Washington

Authors' addresses:

D. Joseph: Computer Sciences and Mathematics Departments, University of Wisconsin
1210 West Dayton St., Madison, WI 53706.

P. Young: Computer Science Department FR-35, University of Washington, Seattle, WA 98195.
(1988-89: Brittingham Visiting Professor
Computer Sciences Department, University of Wisconsin
1210 West Dayton St., Madison, WI 53706.)

SELF-REDUCIBILITY:

THE EFFECTS OF STRUCTURE ON COMPLEXITY*

DEBORAH JOSEPH
University of Wisconsin - Madison
PAUL YOUNG
University of Washington

Abstract. In this column we will discuss the effect that various *self-reducibility* properties have on the analysis of complexity classes. We will be primarily interested in reviewing some of the more elementary results for readers unfamiliar with the field and then discussing some recent results and directions where self-reducibilities have been useful. Throughout, we will focus on the question of when self-reducibilities can be used to force sets, or classes of sets, to have lower complexity than might otherwise be expected.

Contents.

Categories and Subject Descriptors: F.1.3 [Theory of Computation]: complexity classes
General Terms: Structural complexity theory
Additional Key Words and Phrases: self-reducible, near-testable, p-cheatable

* Work of the first author supported in part by the National Science Foundation under grant DCR-8520597. Work of the second author supported by a Brittingham Visiting Professorship in the Computer Sciences Department at the University of Wisconsin-Madison.

The authors are grateful to Eric Bach, Danilo Bruschi, Anne Condon and Meera Sitharam for useful discussions and comments. This paper will appear in 1988 as a special guest column in the Bulletin of the EATCS.

1. INTRODUCTION.

In this column we will discuss the effect that various *self-reducibility* properties have on the analysis of complexity classes. A set, S, is *self-reducible* if the membership question for any element can be reduced in polynomial time to the membership question for a number of *shorter* elements. The classic example of a self-reducible set is SAT, the set of satisfiable Boolean formulas. SAT is self-reducible because an arbitrary Boolean formula $B(x_1, x_2, \ldots, x_n)$ is satisfiable *if and only if* at least one of the two *shorter* Boolean formulas $B(0, x_2, \ldots, x_n)$ or $B(1, x_2, \ldots, x_n)$ is satisfiabile.

The potential importance of the self-reducibility of SAT and other sets was recognized early, and in the 1970's the property was defined, generalized and investigated by Trachtenbrot, Schnorr, Selman, and Meyer and Paterson, ([Tr-70], [Sc-76], [Sc-79], [Se-79], [MP-79]). Nice reviews of this early work can be found in Balcázar ([Ba-87]), Wagner and Wechsung ([WW-86]), Selman ([Se-88]), and Mahaney ([Ma-82b]), this latter particularly for connections with sparse sets. We recommend these more detailed treatments for readers wanting more complete information about the subject. In this column we will be primarily interested in reviewing some of the more elementary results for readers unfamiliar with the field and then discussing some recent results and directions where self-reducibilities have been useful. Throughout, we will focus on the question of when self-reducibilities can be used to force sets, or classes of sets, to have lower complexity than might otherwise be expected.

In one direction, in work going back to Berman, Fortune, Mahaney, Karp-Lipton, Long, and Yap, one uses the fact that variations of SAT are both self-reducible and complete at various levels of the polynomial time-hierarchy to show how reductions to sparse sets can force various collapses in the polynomial time hierarchy. Of the exponentially many elements of size n, sparse sets, by definition, contain only polynomially many elements. Thus results of this form are a strong indication that the information in complete sets cannot be compressed into a polynomial amount of information. A general overview of work on collapsing hierarchy results and the role of sparse sets appeared in this column in June and October of last year, ([Ha-87]).

In another direction, following up early work by Schnorr, self-reducibilities have been used by Selman and Balcázar to show that all functional versions of NP-complete problems are Turing reducible in polynomial time to their corresponding decision problems. More recently, moving further in the direction of classifying functional problems, the self-reducibility of SAT has been used by Krentel, by Wagner, and by Beigel to show that certain functions can be computed in polynomial time by a machine that makes a small number of oracle queries to SAT.

In still another direction, beginning with work by Selman showing that a set is in P *if and only if* it is both *disjunctively self-reducible* and *p-selective,* one can show that self-reducibility properties which are generally expected to define classes broader than P (polynomial time), work in combination to give precise, and sometimes unexpected, characterizations of P.

2. ELEMENTARY APPLICATIONS.

A set A is said to be *Turing self-reducible* (or just *self-reducible* for short) if there is a polynomial time oracle machine M^A that uses A itself as oracle to recognize A, with the restriction that on inputs of length n all of M^A 's queries are required to have *length* less than n.[1] Obviously, all *polynomially* decidable sets are self-reducible in this sense, and all sets that are self-reducible are decidable in exponential time (time $2^{p(|x|)}$).

If we consider the case of SAT, then the self-reducibility mentioned earlier,

$$B(x_1, x_2, \ldots, x_n) \in SAT \iff [\ B(0, x_2, \ldots, x_n) \in SAT \text{ or } B(1, x_2, \ldots, x_n) \in SAT\ ],$$

yields, in an obvious way, a disjunctive *self-reduction tree.* Begin by placing $B(x_1, x_2, \ldots, x_n)$ at the root of the tree and then each node is given two daughter nodes by fixing the first free variable first at 0 ("false"), and then at 1 ("true"). The depth of this tree is n, which corresponds to the length of the possible satisfying assignments, while the number of leaves in the tree is 2^n, which corresponds to the number of possible satisfying assignments. The Boolean formula $B(x_1, x_2, \ldots, x_n)$ at the root of the tree is satisfiable just if, for each level of the tree, at least one node at that level is satisfiable. Furthermore, since the leaves of the tree have no free variables, but simply have assignments of "true" or "false" to the variables, each leaf of the tree can be tested for satisfiability in linear time.

An exponential time and polynomial space algorithm for SAT can thus be obtained from the tree by doing a depth-first search of the tree and using the fact that the formula, $B(x_1, x_2, \ldots, x_n)$, at the root of the tree is satisfiable just if at least one of the leaves of the tree is satisfied. Furthermore, one can terminate the depth first search at the first leaf which belongs to SAT .

This by itself is *not* very interesting, but in [Be-78] and in [MP-79] Piotr Berman and Meyer and Paterson observed that if SAT is reducible to *any* set by a polynomial time computable function g, then, because the self-reducibility is *disjunctive,* the collisions among the values $g(B_i)$ can be used to *prune* the depth-first search tree: if two nodes of the tree B_i and B_j both map to the same element, i.e., if $g(B_i) = g(B_j)$, then B_i is satisfiable *if and only if* B_j is satisfiable. Thus if B_j lies on a branch to the right of the branch containing B_i *there is no reason to continue the depth first search if the search reaches node* B_j: if B_j is satisfiable the depth first search would already have discovered that node B_i is satisfiable. Thus we can safely *prune* the subtree beginning at node B_j from the depth-first search tree when we reach node B_j. The net result is a depth first search tree for a formula B of length n that has the property that at *any* level of the tree the number of *unsatisfiable* nodes at that level cannot exceed the cardinality of the set $g(\overline{SAT} \cap \{x : |x| \leq n\})$. Thus, if the entire range of g could ever be forced to be *polynomially sparse,* that is, if

$$|\{g(x) : |x| \leq n\}| \leq b(n)$$

for some polynomial b, then the pruned depth-first search tree would have at most $b(n)$ leaves, and so there would be a polynomial time algorithm for SAT. In fact, (Fortune, [Fo-79]), if we merely had that $g(\overline{SAT})$ is

[1] Meyer and Paterson were the first to define self-reducibility as we use it today. To obtain full generality and to preserve the concept under polynomially computable isomorphisms, they measure the "length" of strings by using suitably polynomially computable, polynomially well-founded partial orders. For our purposes here, using ordinary lengths of binary strings is adequate.

sparse, the pruned tree would have at most $b(n)$ *unsatisfied leaves,* and so the depth-first search algorithm would terminate after visiting at most $b(n)+1$ leaves of the pruned tree. Thus, in the case where SAT is reducible by g to a *co-sparse* set, the depth-first search tree can safely be pruned to a polynomial sized tree by pruning nodes which map under g to a common value and by terminally pruning the depth first search tree after it has reached only $b(n)+1$ leaves. Therefore, if $g(\overline{SAT})$ is sparse there is a polynomial time decision procedure for SAT.

For later use, note first that *any* pruning of the self-reducibility tree for SAT to a polynomial sized tree yields a polynomial time decision procedure for *some subset* of SAT, provided our acceptance rule is that we accept the formula at the root of the tree just if at least one of the leaves is verified to be a satisfying instance of the formula. Note also that to work correctly on formulas of length n, g only needs to be a correct reduction for all formulas of length $< n$.

With Fortune's result, we see that, unless $P = NP$, no co-sparse set can be *hard* for NP under polynomially computable *functional* reductions. Now if we instead had that $g(SAT)$ is polynomially sparse, we would know that the pruned depth-first search tree had at most $b(n)$ *satisfied* leaves. Thus, after visiting at most $b(n)+1$ leaves of the pruned tree, the depth-first search algorithm would have visited an *unsatisfied* leaf. But, since a test for *satisfiability* cannot halt when it simply reaches an *un*satisfied leaf, this case does *not* yield a polynomial time algorithm for SAT.

However, this situation deserves closer scrutiny. For this case, Mahaney, ([Ma-82a]), gave a method for "estimating" (or guessing) an "optimal" pruning of the self-reducibility tree. This is a clever technique, which can be expected to have wide applicability. For example, as discussed recently in this column, it is one historical root for the technique used by Immerman (and by Szelepcsényi, see [HR-88]) in their recent independent work proving the closure of various nondeterministic space classes under complementation.

When Mahaney proved that if SAT is $\leq_m^P$ reducible to a sparse set S, then $P = NP$, his *proof* required that S be in NP. In the same paper he pointed out how, given a reduction of a set A in NP to an arbitrary sparse set S, to give a reduction of A to a sparse set $S\#$ *in NP.* However, this seems often to be overlooked, and in the subsequent literature, Mahaney's theorem is frequently misquoted as requiring that the sparse set S be *in NP.* It is thus instructive to modify Mahaney's (and Fortune's) proof(s) to directly obtain the full version of Mahaney's theorem.

Theorem, [M-82a]. *$P = NP$ if and only if there exists a sparse set S such that $SAT \leq_m^P S$.*

For the proof, we begin by considering Fortune's *proof* discussed above. This time however, we have that $SAT \leq_m^P S$ where S rather than $\overline{S}$ is sparse, and instead of bounding $|g(\overline{SAT} \bigcap \{x : |x| \leq n\})|$, the bounding function b now insures

$$|g(SAT \bigcap \{x : |x| \leq n\})| \quad \leq \quad b(n).$$

Following Mahaney, we now define the *pseudo-complement* of SAT as

$$P(\overline{SAT}) =_{def} \{\langle x,t\rangle \ : t \leq b(|x|) \ \&\ \text{there exist distinct } g(y_1)\ldots g(y_t) \\ \&\ \text{for each } i \ \ [y_i \in SAT \bigcap \{x : |x| \leq n\} \text{ and } g(y_i) \neq g(x)]\}.$$

Clearly $P(\overline{SAT}) \in NP$. For each n, and for each $m \leq b(n)$ define

$$P(\overline{SAT})_{n,m} =_{def} \{x : |x| \leq n \text{ and } \langle x, m\rangle \in P(\overline{SAT})\}.$$

Now if $m_0 = |g(SAT) \cap \{x : |x| \leq n\}|$, i.e., if m_0 happens to be the actual number of distinct elements of S which elements of length less than or equal to n in SAT map to under g, then it is easy to see that

$$P(\overline{SAT})_{n,m_0} = \overline{SAT} \bigcap \{x : |x| \leq n\}.$$

I.e., since $P(\overline{SAT})$ is in NP, the "optimal guess," m_0, of the value m "seems" to place $\overline{SAT}$ into NP. Since SAT is complete for NP

$$P(\overline{SAT}) \quad \leq_m^P \quad SAT \quad \leq_m^P \quad S,$$

so let f be a polynomial time computable function which reduces $P(\overline{SAT})$ to S.

Let b' be a polynomial bounding function, derivable from the sparsity of S, from b, and from f satisfying

$$|\{f(x,m) : |x| \leq n \text{ and } m \leq b(n)\}| \ \leq \ b'(n).$$

Next, for any fixed n, for all x of length $\leq n$, consider each of the reductions

$$\lambda x f(x,m) \quad \text{for constants } m \text{ with } m \leq b(|x|).$$

For each of these $b(|x|)+1$ reductions, we can prune the self-reducibility tree for SAT by using the reduction $\lambda x f(x,m)$ to prune the branches of the tree and stopping the generation of the tree once it has $b'(|x|)+1$ leaves. This yields a polynomial number of pruned self-reduction trees, each of polynomial size. So we can test them all in polynomial time. We have already seen that for each n the "optimal" guess m_0, which guesses $|g(\overline{SAT} \bigcap \{x : |x| \leq n\})|$, actually yields (on all formulas of length less than or equal to n) a reduction of $\overline{SAT}$ to the sparse set S. Thus, from the proof of Fortune's theorem, we see that for *this* choice of m the pruned reduction tree must actually decide SAT in polynomial time. While the other pruned reduction trees cannot be expected to decide SAT, we have already remarked that a pruned reduction tree for SAT can never yield an answer "yes" unless the root of the tree really is in SAT. Thus a root element is in SAT *if and only if* at least one of these polynomially many pruned reduction trees claims that it *is* in SAT.

This shows that if SAT is $\leq_m^P$ reducible to a sparse set, then $P = NP$. The proof of the converse is trivial, since in this case SAT would be reducible in polynomial time to *any* set except the empty set and the universal set.

A number of authors, including Ukkonen ([Uk-83]), Yap ([Ya-83]), and Yesha ([Ye-83]), have extended the Fortune-Mahaney collapse of NP to P. For example, Yesha shows that if there is a sparse set S which is *hard* for *co-NP* or *complete* for NP under a *positive* polynomial time bounded-truth-table reducibility then $P = NP$. Given the current interest in the Boolean hierarchy (the closure of NP under polynomial time bounded-truth-table reductions), it would be particularly interesting to know whether the Fortune-Mahaney-Yesha results can be extended to *all* bounded-truth-table reductions. We should note here that Book and Ko, ([BK-87]), have shown that the collection of all sets $\leq_{k-tt}^P$ reducible to a sparse set is properly contained in the collection of all sets $\leq_{k+1-tt}^P$ reducible to a sparse set, and that these separations can be witnessed by sets which are in EXP. An extension of the results of Fortune-Mahaney-Yesha to all *bounded-truth-table* reductions would show that the Book-Ko separations can not be accomplished by sets which are in the Boolean hierarchy. (Further separation results for classes reducible to sparse sets may be found in [Ko-88].)

Obviously, the Fortune-Mahaney results apply to sets which are *hard* for *NP* or for *co-NP* under $\leq_m^P$. But another interesting open question, pointed out by Mahaney in [Ma-82b], is the extent to which the Fortune-Mahaney results apply to sets *in* $NP - P$ which are *not* complete for *NP*. The Fortune proof will clearly apply to *any* set which is disjunctively self-reducible, but Mahaney's proof seems to require *both* disjunctive self-reducibility *and* completeness of the set reducible to the sparse set. As Mahaney points out, *GRAPH-ISOMORPHISM* remains a particularly interesting problem. This problem is easily seen to be disjunctively self-reducible. As shown in [MP-79] (and as follows directly from Fortune's proof), the *complement* of *GRAPH-ISOMORPHISM* cannot be reduced to a sparse set without collapsing *GRAPH-ISOMORPHISM* to *P*. But because it is not known whether *GRAPH-ISOMORPHISM* is complete, in spite of its self-reducibility it still remains unknown whether *GRAPH-ISOMORPHISM* can be reduced to a sparse set.[2]

It is easy to see that every disjunctive self-reducible set is in *NP*, ([Ko-83], so it is natural to ask whether *every* set in *NP* is disjunctively self-reducible. However, this is probably not true. In ([Se-88]) Selman shows that if, for linear exponential time, E, $E \neq NE$, then $NP - P$ contains a p-selective set. But, as we shall see later, any disjunctively self reducible set which is $\leq_m^P$ equivalent to a p-selective set is in P.

The above work of Mahaney *et.al.* does not answer the question of whether sparse and co-sparse sets can be complete for *NP* under polynomial time Turing reductions. Various people have worked on this problem, ([KL-80], [Lo-82b], [Ma-82a], [Ya-83]). Perhaps the strongest of these results is by Long, who proves the very elegant (and usually not fully quoted) result that if $SAT \leq_T^P S$ for a sparse set S, *then every set in the polynomial time hierarchy is* $\leq_T^P$ reducible to S. It follows for example that if *SAT* is $\leq_T^P$ reducible to a sparse or co-sparse set in Δ_2^P, then the whole polynomial time hierarchy collapses to Δ_2^P. There is a good discussion of similar results in [Ma-82b]. Mahaney also gives a nice discussion of how the disjunctive self-reducibility of *SAT* can be used to give a Π_1^P description of polynomial sized *circuits* for *SAT* and an excellent discussion of applications of this technique to many results on collapsing the polynomial-time hierarchy. Mahaney also points out that not all of the original Karp-Lipton methods, ([KL-80]), use the technique of self-reducibility, and he raises the question of whether self-reducibility can, in effect, be made a universal method.

Balcázar, Book, and Schöning have worked on this problem, and the results in [BBS-86], [Ba-87], [Ba-88] show that essentially all known results proving that reducibilities of complete sets to sparse sets force collapses in the polynomial time hierarchy can be obtained as a fairly *uniform* application of the self-reducibility technique.[3]

[2] Further work on *GRAPH-ISOMORPHISM,* making use of both self-reducibilities and *promise-problems,* may be found in [Se-88b].

[3] In [Ba-87] Balcázar introduces the notion of *word decreasing query* (wdq) self-reducibility. A set A is wdq self-reducible if there is a polynomial time oracle machine M^A that uses A itself as oracle to recognize A, with the restriction that on an input x all of M's oracle queries must either be to strings shorter than x or to lexicographic predecessors of x. Balcázar observes that all wdq self-reducible sets are decidable in exponential time and shows that if a wdq self-reducible set is in *Pspace/poly*, then it is in *Pspace*. He then uses wdq self-reducible sets to reprove the Karp-Lipton results on the collapse of *EXPTime* that were originally proved by the "round-robin tournament" method.

3. SELF REDUCIBILITY AND FUNCTIONAL COMPUTATION.

Many problems which we traditionally think of as set recognition problems are more naturally thought of as functional computations, or clearly involve functional computations as the most obvious means of solution. For problems like the Traveling Salesperson Problem, this is surely obvious from its statement, but even in other problems, for example clique problems, one is presumably more interested in knowing not just, e.g., whether there is a clique of a given size, but in knowing an actual instance of a maximal clique, or perhaps the maximum size of a clique. Even in asking whether the maximal clique of a graph is unique, which is a set-recognition problem, one expects that the difficulty of answering the question should have something to do with how hard it is to find a maximal clique.

In fact, Papadimitriou and Yannakakis, ([PY-82]), show that the problem "does the maximal clique have size *exactly k?*" is complete for the class D_2^P which is at the second level of the *Boolean* hierarchy over NP, and (presumably) well below Δ_2^P, which is at the second level of the polynomial time hierarchy. While showing that a problem is complete or *hard* for NP is generally believed to be adequate for showing that no fast (i.e., polynomial time) algorithms exist for the set, knowing the exact classification and structure of such sets is important because, as discussed by Papadimitriou and Yannakakis, ([PY-88]), such results may have a bearing on whether efficient *approximation* or *probabilistic* algorithms will exist for the sets. As we shall see, Krentel (and others) have recently shown that understanding hierarchies of *functional* computations may yield useful insight into such *set* classification problems.

While mathematically we are trained to think of functions as being reducible to sets, (namely each function is equivalent to its *graph),* from the point of view of computations this notion of equivalence seems too gross. For example, given a method of computing a function, one can reduce the question of whether a pair $\langle x, y\rangle$ is in the graph of the function by making *a single* request for a computation of f. But the reverse question, given x asking what is the *value* of $f(x)$, is by no means easily related to asking membership questions about the graph of f. In the worst case, one might need $f(x)$ queries to the graph to compute $f(x)$. Even if one is clever and tries to code a function f, not by the graph of f, but instead by the more complicated set,

$$S_f =_{def} \{\langle x, y\rangle : y \text{ is an initial segment of } f(x)\},$$

it still seems to require $|f(x)|$ questions to S_f to compute $f(x)$.

It is interesting to observe that many of the functions one is interested in computing, for example, the minimal traveling salesperson tour or the maximal satisfying assignment, are self-reducible as *functions.* To date, this fact, the self-reducibility of *functional* computations, seems to have had little direct use in the literature. However, as we shall see, the self-reducibility of sets like SAT has been well-exploited in building *functional* hierarchies which help in analyzing set recognition problems.

Trachtenbrot, ([Tr-70]), in his early work on *auto-reducibility,* was interested, not directly in the self-reducibility of *sets,* but rather in the question of when, in computing a function $f(x)$, knowing values of $f(y)$ for $y \neq x$ would help in the computation of $f(x)$. Because his definition did not require $y < x$, it led, not to the notions of self-reducibility which we have been exploring here, but rather to questions about *helping,* a broad topic with many new and interesting results,[4] and to questions about *p-cheatability.* However

[4] We do not have space in this column to discuss the many new results relating *robust* oracle machines, *helping* and self-reducibilities, but we encourage the reader to look at [Sc-88], [Ba-87], [Ko-87] and [HH-87] to gain a feeling for this rich area of research.

by the mid to late 1970's, Schnorr was interested in using self-reducibilities to analyze the difficulty of functional computations, ([Sc-76], [Sc-79]). He showed that for sets in NP which have a certain special form of disjunctive self-reducibility, there is a related functional problem for which the functional computation is "equivalent in difficulty" to the original decision problem.

But as we have already seen, it is not likely that every set in NP is disjunctively self-reducible. Nevertheless, it is natural, to associate with any set A in NP the set of all initial segments of the strings which "verify" that elements of A are in A, (for example, the initial segments of satisfying assignments for Boolean formulas). This set is obviously self-reducible, and one might expect it to be somehow equivalent to A. But Selman, ([Se-84]) gives a proof, credited to Borodin and Demers, ([BD-76]), which shows that if $P \neq NP \cap coNP$, then there is a set $A \in NP - P$ for which the associated set of "verifying prefixes" is not even Turing reducible to A in polynomial time. In [Se-88] Selman improves this to show that if $P \neq NP$, then there is a set $A \in NP - P$ for which the associated set of "verifying prefixes" is $\leq_T^P$ *−complete* for NP *even though* A itself is *not* complete for NP. On the other hand he proves that if A *is* $\leq_T^P$ *−complete* for NP, then A is polynomial Turing equivalent to its set of "verifying prefixes." In [Ba-87], generalizing Schnorr's results on the equivalence of certain NP problems and their functional counterparts to *all* NP -complete problems, Balcázar uses the self-reducibility of initial segments of verifying solutions in much the same way. For example, he shows that for an arbitrary NP-complete set A, a function, $f(x)$, *verifying* that $x \in A$ can always be computed in polynomial time *relative to an oracle for* A.

Balcázar's interpretation of this result is that for NP-complete problems there is always a functional solution which is "not much harder to compute than A itself." Obviously, except for the polynomial bound on the number of queries, this interpretation ignores the number of queries which the functional computation must make to the oracle, A.

Recently there has been a lot of work developing hierarchies of functions which are based on the number of queries which the computations make to oracles. Some of this, for example the work of Beigel, ([Be-87c]), shows that such hierarchies often differ significantly from the hierarchies of sets which are built by the same criteria. A very nice survey covering a wide variety of topics in this area (adaptive versus parallel queries, complete problems for bounded query classes, small numbers of queries versus sparse oracles, polynomial truth-table reducibilities and the Boolean hierarchy) can be found in Wagner's 1988 *Structure in Complexity Theory* paper, ([Wa-88]). Such work demonstrates the importance of studying functional computations in their own right, and not simply as a little understood adjunct of set recognition problems, which are the traditional domain of complexity theory.

Of the published work relating functional computations to set recognition problems, perhaps Krentel's work in [Kr-86] makes the most explicit use of functional hierarchies. In a manner analogous both to the way one defines the Boolean hierarchy over NP and to the way one defines the set $\Delta_2^P =_{def} P^{NP}$ at the second level of the polynomial time hierarchy, for any well-behaved polynomially computable and polynomially bounded function function b, Krentel defines the functional class

$$FP^{SAT}[b] =_{def} \{f : f \text{ can be computed in polynomial time} \\ \text{using at most } b(|x|) \text{ queries to } SAT \text{ for each input } x\}.$$

The union of these classes $FP^{SAT}[b]$ over all polynomial bounds b gives all of the functions which can be computed in polynomial time given a set in NP as oracle, and is thus directly analogous to the set recognition class Δ_2^P. But Krentel shows that these functional hierarchies may be of particular interest when the bounding functions, b, are subpolynomial. The *self-reducibility* of sets like SAT has been well-exploited in the development of this and similar functional hierarchies. As just one example, we will discuss Krentel's work, ([Kr-86]).

With problems in NP, we traditional associate with accepting paths the simple answer "yes," which gives only limited information about the problem. In the case of a problem like *CLIQUE*, we might want to get more information by associating with any accepting path the *size* of the clique (if any) that is found along the path. This gives more information than a simple "yes" answer, but less information than would be supplied by outputting the clique itself. In the case of a problem like *Traveling Salesperson,* one might want to output the length of the tour found by the path, or even the tour itself. And with SAT we can well imagine wanting each successful path to output the satisfying assignment which it finds. In the terminology used in [Wa-86], these output variations amount to changing the *valuation functions,* β_A, which Wagner assigns to the accepting paths of the validations that $x \in A$.

More interestingly, for a problem like *BIN PACKING* for which there is a polynomial time algorithm which approximates the optimal solution to within $O(log^2(|x|))$, ([KK-82]), one might want to output for each path the difference of the path's proposed solution and the calculated polynomial time approximation to the optimal solution. This would enable one to recover the optimal solution from the difference calculated along the path.

Now consider nondeterministic Turing machines which, along each "accepting" computation path produce some functional output value. For any such machine N, define $OptP_N(x)$ to be the largest value produced on any accepting computation path for input x. (Or, for problems like *TSP,* take the smallest value.) Clearly, many problems which we normally associate with nondeterministic set recognition machines can be more crisply formulated as questions about calculating functions in the class, *OptP*, of functions computed in this manor by nondeterministic machines. For any well-behaved, polynomially computable bounding function b, Krentel defines

$$OptP[b] =_{def} \{f : f \in OptP \text{ and } |f(x)| \leq b(|x|)\}.$$

Note that *a priori*, b is *not* a bound on any computational aspect of f; it merely bounds the size of the output. Nevertheless, because of the disjunctive self-reducibility of SAT, it is easy to establish a tight relationship between the classes $OPT[b]$ and the functional hierarchies $FP[b]$:

Definition, [Kr-88]. *For functions f and g, define $f \leq^P_{1-tt} g$ if there exist polynomially computable functions, T_1 and T_2 for which $f(x) = T_2(x, g(T_1(x)))$.*[5]

Using the obvious self-reducibility of the functions in *OptP*, it is not difficult to prove that, ([Kr-86]), up to $\leq^P_{1-tt}$ reductions, the classes $OptP[b]$ and $FP^{SAT}[b]$ are the same for each b. (The full power of $\leq^P_{1-tt}$ reductions are not used here.) Furthermore, for functions f in *OptP* and $\leq^P_{1-tt}$ reductions there is a tight relation between the functional query hierarchies and more traditional set hierarchies. For example:

[5] Krentel calls $\leq^P_{1-tt}$ a *metric* reduction, presumable because it will preserve the number of queries in oracle computations. It is also clearly the correct generalization of *one bounded-truth-table* reductions in the functional context.

Theorem, [Kr-86].

1. *If f is complete for OptP, then there is a natural polynomially time computable predicate P such that $P(x, f(x))$ is complete for Δ_2^P.*
2. *If f is hard for OptP, then $\{\langle x, k\rangle : f(x) = k\}$ is complete for D^P.*
3. *If f is hard for OptP, then $\{\langle x, k\rangle : f(x) \geq k\}$ is complete for NP.*

The following are examples of functional problems which Krentel shows to be complete for *OptP*: traveling salesperson, lexicographically maximal satisfying assignment, value of knapsack solutions. For *OptP(log)* Krentel shows the following functional problems to be complete: maximal number of satisfying assignments, size of largest clique, the chromatic number of a graph, the longest cycle in a graph, etc.

Theorems like the preceding can thus be used to project naturally occurring *functional* problems onto set recognition problems which occur above *NP* in the polynomial time and Boolean hierarchies. We refer the reader to [Kr-86] and [Wa-86] for further details.

Using the Karmarkar-Karp approximation algorithm for *BIN PACKING* discussed earlier, the optimal number of bins can be approximated to within $O(log(log))$ bits, which thus places the problem of calculating the optimal number of bins in *OptP(log(log))*. Whether this problem is complete for this class is unknown. However, in view of the following theorem, it cannot be complete for $OptP(g)$ for $g \geq log$ without collapsing *NP* to *P* :

Theorem, [Kr-86]. *For honest and monotone functions f and g with $f < g$ and with $f \leq (1 - \epsilon) \cdot log$, if $FP^{SAT}[f] = FP^{SAT}[g]$ then $P = NP$.*

The proof of this theorem is obtained by exploiting the disjunctive self-reducibility of the optimal solution of *SAT* . Observe that for any Boolean formula $B(x_1, \ldots, x_n)$ the function which on input B gives the first $g(n)$ bits in the maximal satisfying assignment to $x_1, \ldots, x_n$ is a function in $FP^{SAT}[g]$. Thus it can be calculated by an $FP^{SAT}[f]$ machine. But since $f < log$, we can simulate the results of the answers to all possible oracle queries in time polynomial in $|x|$ *without actually querying the oracle.*

Simulating the $FP^{SAT}[f]$ machine allows us to produce a new Boolean formula

$$B'(y_1, \ldots, y_{f(n)}, x_{g(n)+1}, \ldots, x_n, x_1, \ldots, x_{g(n)})$$

such that *any* truth assignment to the variables $y_1, \ldots, y_{f(n)}$ forces a unique assignment to $x_1, \ldots, x_{g(n)}$.[6] This process may be iterated repeatedly until an equivalent formula is obtained which has at most $log(n)$ truly "free" variables. Once this is achieved, we can substitute the n possible satisfying assignments to these $log(n)$ variables, into *this* formula. For each of these possible satisfying assignments, we can calculate the required values for the remaining variables, test the resulting assignment to see if it satisfies the formula, and from these n possible satisfying assignments read off the maximal satisfying assignment. This gives a polynomial time decision procedure for *SAT*.

Krentel's theorem, which (modulo $P \neq NP$) establishes that the $FP^{SAT}[b]$ query hierarchy is a true functional hierarchy up to logarithmic bounds, lends further insight into why problems such as finding the

[6] Note that, while this process does not reduce the actual *size* of the formulas, in the larger sense it is still a disjunctive self-reduction since it does produce a formula which is smaller in the partial ordering of formulas obtained by *counting only the free variables.*

size of the largest clique, which is complete for $FP^{SAT}[log]$, and finding the length of an optimal traveling salesperson tour, which is complete for FP^{SAT}, should indeed be viewed as having increasing levels of difficulty. And, as pointed out by Krentel, this functional hierarchy makes the problem of characterizing the optimal number of bins for *BIN PACKING,* which lies in $FP^{SAT}[loglog]$, particularly intriguing.

We should point out that using similar methods Wagner, ([Wa-86]), has also obtained many results like those above, and he does so in a nicely systematic fashion. For our purposes, we have concentrated on Krentel's exposition simply because it makes more explicit use of functional methods and gives a particularly nice result about functional hierarchies.

4. CHARACTERIZING POLYNOMIAL TIME.

In the preceding sections we discussed results which show how the self-reducibility of *SAT* can be used to prove that certain problems cannot be solved with only a polynomial amount of information unless $P = NP$ or some other collapse occurs in the polynomial time hierarchy.

In this final section, we will consider generalizations of the notion of self-reducibility which can be used to characterize polynomial time. Our starting point is two early results of Selman, ([Se-82b]), which show that P can be characterized as those sets which are both disjunctive self-reducible and p-selective, and also as those p-selective sets which are positive truth-table reducible to their complements.

Definition, [Se-79]. *A set S is said to be p-selective if there is a polynomially computable function s which, given elements x and y always computes $s(x,y) \in \{x,y\}$, and $s(x,y) \in S$ if and only if at least one of x and y is in S.*[7]

Thus, from x and y, s is guaranteed to select "a most likely candidate" for membership in S. For example, the set of optimal solutions of the the Traveling Salesperson Problem is a p-selective set, as are most other optimal solutions of *NP*-complete problems. Ko, ([Ko-83]), has characterized the p-selective sets as those sets, S, for which there is a polynomially computable *preordering* for which S is a Dedekind left cut in the induced linear ordering.

Although p-selective sets are not Turing self-reducible in the sense defined earlier, p-selectivity is a weak form of the *intuitive* notion of self-reducibility since it relates the membership question for any *two elements* to the question of membership for the smaller of the two elements (in some suitable ordering).

Here we are interested in Selman's proof that one can characterize the polynomially decidable sets, P, as those sets which are both disjunctive self-reducible and p-selective. To see this, simply observe that if a set, S, is p-selective, then we can prune the self-reducibility tree as follows: in a linear pass over the daughters of the elements at any node (doing a *breadth first search*) we can find an element "most likely" to be in the set S. Since the self-reduction tree is disjunctive, we can prune all daughters except this most likely daughter from the tree. Continuing in this fashion, in a polynomial number of steps at each level of the self-reduction tree, we can prune the entire self-reduction tree to a "telephone pole." The root node is then in the set if and only if the single leaf of this pruned tree is in S.

[7] The notion of p-selective comes directly from the classical notion of "semi-recursive" studied by Jockusch in early papers in classical recursion theory, e.g. ([Jo-68]). P-selectivity and other variants of self-reducibility are discussed in the survey, [GJY-87a], which has very little overlap with the material discussed in this column. We recommend the technical report version of [GJY-87a], which was distributed at the conference, and which is more complete than the Proceedings version. Copies are available from either the University of Washington or the University of Wisconsin Computer Science Departments.

Recently, working with Judy Goldsmith we have used variants of Balcázar's notion of word-decreasing-query self-reducibility, p-cheatability, and self-reducibility to give new characterizations of the class of sets decidable in polynomial time.

Definition, [GJY-87a]. *A set A is near-testable if there is a polynomially computable function which, given $w \neq 0$, decides whether exactly one of w and $w-1$ is in A, where $w-1$ is the lexicographic (or numerical) predecessor of w. That is, the function*

$$f(w) = \chi_A(w) + \chi_A(w-1) \ (mod\ 2)$$

is computable in polynomial time.

For later use notice that if a set is near-testable, then it is word-decreasing-query (wdq) self-reducible with $w-1$ being the only query necessary to answer membership for input w. Obviously, all wdq self-reducible sets that require only one query to the set, including all near-testable sets, are decidable in polynomial space. We originally became interested in studying near-testable sets because they are an especially easily defined class of sets, which lie someplace between P and *Pspace,* but require no appeals to "complicated" concepts such as nondeterminism or randomness for their definition.

The other variant of self-reducibility we will be interested in is the notion of *p-cheatability.* The *p-cheatable* sets were first studied by Amir, Beigel, Gasarch, Gill, Hay and Owings, ([AG-87], [Be-87a], [Be-87b], [BGGO-86], [BGH-87], [BGO-87]). The definition of the class is based on the following scenario: one is given a large collection of inputs, $w_1, w_2, ..., w_n$, and one would like to determine whether or not each input is in a known set A. It is assumed that the membership problem for A is difficult. Therefore, if knowing whether some w_i is in A helps in determining whether another w_j is in A, then one would like to use this information. These authors use this idea to motivate the following machine model. To determine membership in A, one designs an oracle machine M that, when given A as oracle will determine membership in A for n different inputs by asking the oracle as few questions and doing as little computation as possible. For example, one might hope to design machines that run in polynomial time and for a *fixed* k decide membership for 2^k inputs by asking only k questions. The definition can be formalized as follows.

Definition, [AG-87], [Be-87a]. *A set A is (n for k) p-cheatable if there exists a polynomial time oracle machine M using A as oracle such that, if M^A is given inputs $(w_1, ..., w_n)$, then, with k or fewer queries to the oracle, M^A determines membership in A for each of $w_1, ..., w_n$. (We will be primarily interested in the case where $n = 2^k$ and in this column we will often call (2^k for k) p-cheatable sets simply p-cheatable.)*

Note that the queries that M asks in the course of deciding membership for $w_1, ..., w_{2^k}$ are *not* restricted to come from the set $\{w_1, ..., w_{2^k}\}$. In fact, for our purposes the exact questions asked and the oracle to which they are addressed turn out not to be relevant.[8] It will turn out that the only thing that matters is the *number* of questions that are asked. Notice that k is a constant that will depend on the set A.

[8] Beigel's definition of p-cheatable sets (sets which we call (2^k for k) p-cheatable) allows M access to *any* fixed oracle. However, the definitions given by Amir and Gasarch for ($2^k - 1$ for k) p-cheatable sets require that M use A as the oracle. (These sets are called *verbose* by Amir and Gasarch.)

To put the results that follow in context it is useful to observe the following.

- Amir and Gasarch ([AG-87]) have shown that there are p-cheatable sets of arbitrarily high time complexity. Thus, there are p-cheatable sets that are not wdq self-reducible, and thus not near-testable.
- In [GJY-87a] we have shown that if one-way functions exist, then there are near-testable sets not in P. In fact, in a weak sense, the class of near-testable sets turns out to be $\leq_m^P$ equivalent to the Papadimitriou-Zachos class, *parity-P*, ([GHJY-87]).

Thus it is highly likely that both the near-testable sets and the p-cheatable sets are proper extensions of the class P of sets decidable in polynomial time.

While the analysis of (2^k for k) p-cheatable sets for arbitrarily large (but) fixed k often involves more difficult combinatorial arguments, one frequently, but not always, gains the necessary fundamental insight for analyzing the general case by considering the special case $k = 1$. For example, in the case $k = 1$, if we are given two elements a and b and we consider any algorithm which decides both $a \in S$ and $b \in S$ by making only one oracle call, then without consulting the oracle we can ask what the outcomes would be if the oracle returned the answer "yes" *and also* if the oracle returned the answer "no." Now one possibility is that independent of whether the oracle answers "yes" or "no" the algorithm decides, for example, that $a \in S$. In this case, in polynomial time we will have learned that one of the pair, namely a, is in S. What happens if the algorithm does not locate either a or b in S or in $\overline{S}$? Then the algorithm must yield different answers to the question "$a \in S$?" depending on whether the oracle answers "yes" or "no." And in this case the same thing must happen for b. If the patterns for a and for b agree, then we know that $a \in S \iff b \in S$, while if the patterns for a and for b disagree, then we know that $a \in S \iff b \notin S$. Summarizing, if S is (2 for 1) p-cheatable, then there is a polynomial time algorithm which, (without the use of an oracle), given any a and b either places at least one of a or b in S or in $\overline{S}$, or else the algorithm tightly binds the membership questions of a and b.[9]

In the case where a and b are adjacent elements, this relationship is very close to the near-testability relationship. But there is a significant difference. Suppose we knew *a priori* that $a \in S$. Then the near-testability relationship would enable us to decide whether $b \in S$, but if we applied the cheatability relationship then we might simply again get the information that $a \in S$, which by itself is of no help in determining whether $b \in S$.

The next result shows that, in spite of the fact that it is unlikely that either can be used alone to characterize polynomial time, near-testability and p-cheatability *can* be combined to characterize polynomial time.

Theorem, [GJY-87c]. *For any fixed k, a set S is in P if and only if S is both near-testable and (2^k for k) p-cheatable.*[10]

If S is in P, the result is of course trivial. For the converse, we prove the result here only for $k = 1$, referring the reader to [GJY-87c] for the proof for arbitrary k.

[9] The result in this paragraph (for $k = 1$) appears in [GJY-87a] and as Lemma 5.4.9 of [Be-87b]. An outline of the corresponding result for arbitrary k appears in [GJY-87a] and in full detail as part of the proof of Theorem 1 in [GJY-87c].

[10] Richard Beigel has informed us that he independently obtained this theorem for the special case $k = 1$, (personal communication).

The idea is as follows. Suppose we have $t+1$ integers, $a_0 < a_1 < \ldots < a_t$, and we know whether or not $a_0 \in S$ and furthermore for each k, $(0 < k < t)$, we have a relation which, given the answer to $a_k \in S$ provides the answer to $a_{k+1} \in S$. We call the interval from a_0 to a_1 the *critical region* because if we could determine a relationship which, given the answer to $a_0 \in S$ would determine a_1, we would then know the membership of *all* of the a_i. Obviously, if $a_1 = a_0 + 1$, we could use the *near-testability* of S to determine such a relationship. This will give us the *final* step of our algorithm. The algorithm for testing membership in S now goes as follows:

To decide whether an element $b \in S$, begin by setting $a_0 = 0$ and $a_1 = a_t = b$. This makes the interval from 0 to b the critical region. Proceeding recursively then, if $a_1 = a_0 + 1$, we use near-testability on a_0 and a_1 and we are done. Otherwise, let c be the midpoint between a_0 and a_1. Apply the p-cheatability algorithm to the pair c and a_1. Three things can happen.

1. If the p-cheatability algorithm tells us whether or not $a_1 \in S$, we are done, for we can then successively find the answers to membership for each a_i.
2. If the p-cheatability algorithm tells us whether or not $c \in S$, we can reset $a_0 = c$ and try again. In this case we have cut the length of the critical region by half.
3. If the p-cheatability algorithm tells us how the membership question for $c \in S$ relates to the membership question for $a_1 \in S$, we can insert c between a_0 and a_1, making the interval from a_0 to c the new critical region. To do this we simply relabel each a_i as a_{i+1} for all $i \geq 1$ and then reset a_1 to c. Note that this again cuts the length of the critical region by half.

Since at each basic iteration of the algorithm, either the algorithm terminates or we cut the size of the critical region by half, the total number of iterations cannot exceed $log(b)$. It is thus easy to see that this method gives a polynomial time algorithm for S. The idea for $k > 1$ is similar, but the combinatorics are more difficult. We refer the reader to [GJY-87c] for details.

The question of whether this result can be proved with a word-decreasing-query condition replacing the near-testability condition is open, although some partial results may be found in ([GJY-87c]).

In closing, we shall discuss one final characterization of the polynomial time decidable sets using variations of self-reducibilities. We remarked above that for p-cheatable sets, when we reduce the membership question for 2^k elements of the set to k elements of the set then the k elements of the set need not be among the original 2^k elements, or even be of the same or smaller size. Indeed, if given $k+1$ elements we can always reduce their membership question to k or fewer elements of smaller size, then it is not too difficult to see that given *any* truth-table self-reduction tree (disjunctive or not), we could systematically use this form of strong p-cheatability to force the self-reduction tree to have finitely bounded width. (If at any time any level of the tree had more than k nodes we could use p-cheatability to prune the tree back to k *critical* nodes which are shorter than the original nodes, (although possibly not among the original nodes).

What is surprising is that this can be made to work for the most general form of self-reducibility and for (2^k for k) p-cheatability *even when the sizes of the queried nodes increase.* This result, was originally proven by Beigel for polynomially *self-truth-table-reducible* sets, ([Be-87b; Theorem 5.4.6). The result can also be proved for arbitrary Turing self-reducible sets. Here we merely state the result. We refer the reader to ([GJY-87c]) for details of the full proof, or to [Be-87b] for details for the simpler case.

Theorem, ([GJY-87c]). *For arbitrary k, a set is in P if and only if it is both self-reducible and (2^k for k) p-cheatable.*

The question of whether this result can be proved for, say, ($2^k - 1$ for k) p-cheatable is open, although some partial results may be found in ([GJY-87c]).

5. BIBLIOGRAPHY.

[AG-87] A. Amir and W. Gasarch, "Polynomially terse sets," *Proc Second Annual Structure in Complexity Conference*, IEEE Computer Society (1987), 22-27; also *Infor and Comput*, **76** (1988).

[Ba-87] J. Balcázar, "Self-reducibility, (extended abstract)" *Symp Theory Automata Comput*, Springer Verlag LNCS (1987), 136-147.

[Ba-88] J. Balcázar, "Logspace self-reducibility," *Proc* 3^{rd} *Ann Structure in Complexity Conf*, IEEE Computer Society (1988), 40-46.

[BBS-86] J. Balcázar, R. Book, and U. Schöning, "The polynomial time hierarchy and sparse oracles," *J. ACM* **33** (1986), 603-617.

[Be-87a] R. Beigel, "Bi-immunity and separation results for cheatable sets," *Manuscript* (June, 1987), 1-15.

[Be-87b] R. Beigel, "Query-limited Reducibilities," *Stanford PhD Dissertation* (October, 1987).

[Be-87c] R. Beigel, "A structural theorem that depends quantitatively on the complexity of SAT," *Proc* 2^{nd} *Ann Structure Complexity Theory Conf*, IEEE Computer Soc, (1987), 28-32.

[BGGO-86] R. Beigel, W. Gasarch, J. Gill and J. Owings, "Terse, super-terse and verbose sets," *University of Maryland Technical Report*, TR-1806 (1986), 1-25.

[BGH-87] R. Beigel, W. Gasarch, L. Hay, "Bounded Query Classes and the Difference Hierarchy," *University of Maryland Technical Report*, TR-1847 (1987), 1-26.

[BGO-87] R. Beigel, W. Gasarch, and J. Owings, "Terse sets and verbose sets," *Recursive Function Theory: Newsletter* **36** (1987), 13-14.

[Be-78] P. Berman, "Relationship between density and deterministic complexity of NP-complete languages," *Symp Math Found Comput Sci, Springer Verlag LNCS,* **62** (1978), 63-71.

[BK-87] R. Book and K. Ko, "On sets reducible to sparse sets," *Proc Third Annual Structure in Complexity Conference*, IEEE Computer Society (1987), 147-154.

[BD-76] A. Borodin and A. Demers, "Some comments on functional self-reducibility and the NP hierarchy," *Dept of CS Tech Report* **TR76-284** *Cornell U.*, (1976).

[Fo-79] S. Fortune, "A note on sparse complete sets," *SIAM J Comput* **8** (1979), 431-433.

[GJY87a] J. Goldsmith, D. Joseph and P. Young, "Self-reducible, p-selective, near-testable, and p-cheatable sets: the effect of internal structure on the complexity of a set, preliminary abstract," *Proc* 2^{nd} *Ann Structure Complexity Theory Conf* (1987), 50-59. Also in more complete form as *U Washington Tech Report, #* **87-06-02,** and as *U Wisconsin Tech Report, #* **743,** 1987, 1-22.

[GJY87b] J. Goldsmith, D. Joseph, and P. Young, "A note on bi-immunity and p-closeness of p-cheatable sets in P/poly," *U Washington Tech Report, #* **87-11-05,** and *U Wisconsin Tech Report, #* **741,** (1987), 1-13, *J Comput Sys Sci,* to appear.

[GJY87c] J. Goldsmith, D. Joseph, and P. Young, "Using self-reducibilities to characterize polynomial time," *U Washington Tech Report,* # **87-11-11**, and *U Wisconsin Tech Report,* # **749**, (1987), 1-20, *Information and Comput,* submitted.

[GHJY-87] J. Goldsmith, L. Hemachandra, D. Joseph, and P. Young, "Near-testable sets," *U Washington Tech Report # 87-11-06* (1987), 1-22. *SIAM J Comput,* submitted.

[Ha-87] J. Hartmanis, "The Structural Complexity Theory Column," *Bulletin EATCS,* **32** & **33** (1987), 73-81 & 28-39.

[HH-87] J. Hartmanis and L. Hemachandra, "One-way functions, robustness, and the non-isomorphism of NP-complete sets," *Proc* 2nd *Ann Structure Complexity Theory Conf,* IEEE Comput Soc (1987), 160-174.

[Hr-88] J. Hromkovič, "Two independent solutions of the 23 years old open problem in one year, or *NSpace* is closed under complementation by two authors," *Bulletin EATCS* **34** (1988), 310-312.

[Im-88] N. Immerman, "Nondeterministic space is closed under complementation," *Proc* 3rd *Ann Structure Complexity Theory Conf,* IEEE Computer Society, (1988), 112-115.

[Jo-68] C. Jockusch, Jr., "Semirecursive sets and positive reducibility," *Transactions of the AMS* **131** (1968), 420-436.

[Ka-88] J.Kadin, "The polynomial time hierarchy collapses if the Boolean hierarchy collapses," *Proc* 3rd *Ann Structure Complexity Theory Conf,* IEEE Comput Soc, (1988), 278-292.

[KK-82] N. Karmarkar and R. Karp, "An efficient approximation scheme for the one dimensional bin-packing problem," *Proc ACM Symp Theory Comput,* **23** (1982), 312-320.

[KL-80] R. Karp and R Lipton, "Some connections between nonuniform and uniform complexity classes," *Proc. 12th ACM Symp Theory Computing* (1980), 302-309.

[Ko-83] K. Ko, "On self-reducibility and weak P-selectivity," *J Computer System Sci* **26** (1983), 209-221.

[Ko-87] K. Ko, "On helping by robust oracle machines," *Proc* 2nd *Ann Structure Complexity Theory Conf,* IEEE Computer Society (June, 1987), 182-190.

[Ko-88] K. Ko, "Distinguishing bounded reducibilities by sparse sets," *Proc Third Annual Structure in Complexity Conference*, IEEE Computer Society (1988), 181-191.

[Kr-86] M. Krentel, "The complexity of optimization problems," *Proc 18th Ann ACM Symposium Theory Computing* (1986), 60-76, *J Comput System Sci,* **36** (1988), to appear.

[La-75] R. Ladner, "On the structure of polynomial time reducibility," *JACM* **22** (1975), 155-171.

[LLS-75]] R. Ladner, N. Lynch, and A. Selman "A comparison of polynomial time reducibilities," *Theor Comput Sci* **1** (1975), 103-123.

[Lo-82]] T. Long, "Strong nondeterministic polynomial-time reducibilities," *Theor Comput Sci* **21** (1982), 1-25.

[Lo-82] T. Long, "A note on sparse oracles for NP," *J Comput Sys Sci,* **24** (1982), 224-232.

[Ma-82a] S. Mahaney, "Sparse complete sets for NP: solution of a conjecture of Berman and Hartmanis," *J Computer Systems Sci* **25** (1982), 130-143.

[Ma-82b] S. Mahaney, "Sparse sets and reducibilities," in *Studies in Complexity Theory,* edited by R. Book, (1982), 63-118.

[MP-79] A. Meyer and M. Paterson, "With what frequency are apparently intractable problems difficult?" *MIT/LCS/TM-126* (1979).

[Pa-82] C. Papadimitriou, "On the complexity of unique solutions," *Proc* 23rd *Annual IEEE Sym Found Comp Sci,* **23** (1982), 14-20.

[PY-84] C. Papadimitriou and M. Yannakakis, "The complexity of facets (and some facets of complexity)" *J Comput Sys Sci,* **28** (1984), 244-259.

[PY-88] C. Papadimitriou and M. Yannakakis, "Optimization, approximation, and complexity classes," *Proc* 20^{th} *Annual ACM Sym Theory Comput,* **20** (1988), 229-234.

[Sc-88] U. Schöning, "Robust oracle machines," *Proc of the Symposium on the Mathematical Foundations of Computer Science,* (1988).

[Sc-76] C. Schnorr, "Optimal algorithms for self-reducible problems," ICALP (1976), ed Michaelson and Milner, *Edinburgh Univ. Press,* 322-337.

[Sc-79] C. Schnorr, "On self-transformable combinatorial problems," Symp on Math. Optimierung, Oberwolfach, 1979.

[Se-79] A. Selman, "P-selective sets, tally languages, and the behavior of polynomial reducibilities on NP," *Math Systems Theory* **13** (1979), 55-65.

[Se-82a] A. Selman, "Analogues of semi-recursive sets and effective reducibilities to the study of NP complexity," *Inform and Control,* **52** (1982), 36-51.

[Se-82b] A. Selman, "Reductions on NP and P-selective sets," *Theoretical Computer Science,* **19** (1982), 287-304.

[Se-84] A. Selman, "Remarks about natural self-reducible sets in *NP* and complexity measures for public key cryptosystems," manuscript (1984), 34 pgs.

[Se-88a] A. Selman, "Natural self-reducible sets," *SIAM J Comput,* **17** (1988), 989-996.

[Se-88b] A. Selman, "Promise problems complete for complexity classes," *Infor and Comput,* **78** (1988), 87-98.

[Tr-70] B. Trachtenbrot, "On autoreducibility," *Dokl Akad Nauk SSSR* **11** (1970).

[Wa-86] K. Wagner, "More complicated questions about maxima and minima, and some closures of NP," *Proc. of 13th ICALP, LNCS* **226** (1986), 434-443.

[Wa-88] K. Wagner, "Bounded query computations," *Proc. of the Third Structure in Complexity Theory Conference* (1988), 260-277.

[WW-86] K. Wagner and G. Wechsung, "Computational Complexity," D. Reidel Publishing Co., (1986), 551 pgs.

[Ya-83] C.K. Yap, "Some consequences of non-uniform conditions on uniform classes," *Theor Comput Sc,* **26** (1983), 287-300.

[Ye-83] Y. Yesha, "On certain polynomial-time truth-table reducibilities of complete sets to sparse sets," *SIAM J Comput,* **12** (1983), 411-425.