Uniformly Hard Languages*

Rod Downey†
Victoria University

Lance Fortnow‡
University of Chicago

November 19, 2001

Abstract

Ladner [18] showed that there are no minimal recursive sets under polynomial-time reductions. Given any recursive set A, Ladner constructs a set B such that B strictly reduces to A but B does not lie in P. The set B does have very long sequences of input lengths of easily computable instances.

We examine whether Ladner's results hold if we restrict ourselves to "uniformly hard languages" which have no long sequences of easily computable instances. Under a hard to disprove assumption, we show that there exists a minimal recursive uniformly hard set under honest many-one polynomial-time reductions.

1 Introduction

When we look at the complexity class NP of problems computable in nondeterministic polynomial time, we usually consider two groups of problems: Those which have efficient solutions (P) and those which are as hard as any other problem in NP (NP-complete). These two groups may not cover all problems, there may be problems that are too hard to be in P but not hard enough to be NP-complete.

If P = NP then all of the NP problems collapse to P and no such incomplete problems occur. Ladner [18] shows this is the only case.

Theorem 1.1 (Ladner) *If $P \neq NP$ then there exist sets in NP that are neither in P nor NP-complete.*

*This paper first appeared in preliminary form in the Proceedings of the 13th IEEE Conference on Computational Complexity. Much of the research of this paper occurred while both authors were visiting Cornell University.

†Partially supported by New Zealand Marsden Fund for basic science via grant 95-VIC-MIS-0698 under contract VIC-509. Address: School of Mathematical and Computing Sciences, Victoria University, P. O. Box 600, Wellington, New Zealand. Email: downey@math.vuw.ac.nz. http://www.mcs.vuw.ac.nz/people/Rod-Downey.shtml.

‡Supported in part by NSF grant CCR 92-53582. Current Address: NEC Research Institute, 4 Independence Way, Princeton, NJ 08540, USA. Email: fortnow@research.nj.nec.com. http://www.neci.nj.nec.com/homepages/fortnow.

While Theorem 1.1 is widely quoted, the proof is not necessarily widely known. We give two distinct proofs of this result in Appendix A.

These proofs give much more than Theorem 1.1. Polynomial time reducibilities calibrate languages into equivalence classes of the same level of complexity. The equivalence classes are called *degrees*. One example of a degree is the collection of NP-complete languages. The polynomial time degrees form a partial ordering where $\mathbf{a} \leq \mathbf{b}$ iff there is a polynomial time procedure Φ and languages $A \in \mathbf{a}, B \in \mathbf{b}$ such that Φ reduces A to B. The proofs in Appendix A can be easily modified to show that the polynomial time degrees of computable languages are a dense partial ordering.

Theorem 1.2 (Ladner) *For every computable language B not in P, there exists a language A such that*

1. *A is not in P*
2. *A polynomial-time many-one reduces to B*
3. *B does not polynomial-time many-one reduce to A.*

In the proof given in Appendix A.1, as with most reductions arising from concrete problems, the reduction from A to SAT is honest, in the sense that the size of the query is the same as that of the string. (Here, it is $x \in A$ iff $f(|x|)$ is even and x is in SAT.) Thus we remark that the generalization of Ladner's theorem, proven by this method, actually establishes the following.

Theorem 1.3 (Ladner) *For every computable language B not in P, there exists a language A such that*

1. *A is not in P*
2. *A polynomial-time honest many-one reduces to B*
3. *B does not polynomial-time honest many-one reduce to A.*

In the crudest terms, the proof in Appendix A.1 works by cutting holes out of SAT, the language of satisfiable Boolean formulae. The remaining language A contains enough of SAT to be hard but enough holes to not be NP-complete. The length of the holes may be very long to give the language enough time to "look back" to see when a requirement is fulfilled.

The proof is in some sense unsatisfying. At the point of view of some input length, the language A either looks very hard (like SAT) or very easy (the empty set). The candidates we have for natural incomplete problems, factoring, graph isomorphism, discrete logarithm, for example, probably do not have this property. Rather they are "uniformly" harder than P and not NP-complete.

Can we use the techniques of Ladner to create these kinds of incomplete sets? We give evidence against this possibility.

To understand the issue, we first define a notion of "uniform hardness" where a language B is *uniformly hard* if for every A in P there are not arbitrarily large polynomial input ranges where B and A agree.

The language A created by the construction in Appendix A.1 is not even close to uniformly hard. Note that the proof in Appendix A.2 also is not uniformly hard: Since f(n) grows greater than any fixed polynomial then $f(n) - f(n-1)$ also does and there are arbitrarily polynomially large input ranges in L which are empty. However, in the proof from Appendix A.2 the reduction given from L to SAT is not honest.

These observations inspires the following conjecture.

Conjecture 1.4 *If there exist uniformly hard sets in NP then there exists incomplete uniformly hard sets.*

We show that under a hard to disprove assumption there exists minimal recursive uniformly hard sets under polynomial-time honest many-one reductions. This shows that Ladner's proof techniques must produce sets with long sequences of easy instances. Finding nonuniform incomplete sets, at least for honest reductions, will require new techniques.

1.1 An Historical Perspective

The realization that virtually all combinatorial reductions arising from concrete problems were honest (such as those of Karp [17] and Garey and Johnson [12]) came very early, and strong forms of this observation gave rise to conjectures such as the Isomorphism Conjecture of Berman and Hartmanis [8].

Examining the structure of (the degrees of) computable sets under various polynomial-time reductions has a long history. Ladner [18] has the fundamental paper in this area, demonstrating considerable richness in the structures, showing that, for instance for honest and standard polynomial time $m-, tt-$, and $T-$ reducibilities, there were minimal pairs and that the structure was dense. In that paper Ladner introduced what has proven to be the fundamental technique of "delayed diagonalization" which in many instances is applied by making a language essentially trivial "long enough" to cause some desired diagonalization. For instance, in the density theorem, as we observed above, Ladner's proof works by creating a set B which alternates between looking like A and looking like some set in P such as $\emptyset$. These alternations may take a very long time and thus cause B to look like an easy set for very long sequences of input lengths. This technique is ubiquitous in the area. For instance, see Balcázar et. al. [7], Balcázar and Diaz [6], Landweber et. al. [20], Chew and Machtey [10], Homer [14], Regan [23], Schöning [24], and Ambos-Spies [1, 3]. It is even the case that most applications of the so-called "speedup technique" have at their heart a Ladner-type strategy. We refer to, for instance, Downey [11], Shinoda-Slaman [25], Ambos-Spies [2], or Shore-Slaman [26].

One of the key motivations for the study of these structures is to give insight into the natures of computation, nondeterminism, and feasibility. From this point of view much of this work can have a somewhat unsatisfactory flavor. Wolfgang Maass coined the phrase "punching large holes in

sets." In terms of concrete complexity of natural problems, it seems unsatisfactory to say that if P $\neq$ NP then there are problems of intermediate degree because we punch large holes in the language simply because we can exploit the guaranteed totality of the reductions. (The point is that precisely that the arguments given in Appendix A both work in the case of the classical truth table degrees above the computability-theoretical degree of the halting problem.) Often one feels that one is doing elementary set theory rather than complexity-theory. In this light the first motivation of the present paper is to ask what the universe looks like is we only restrict our attention to languages where we won't punch large holes in the languages.

A second related motivation comes from the point of view that the current evidence suggests that not only does P $\neq$ NP, but *hard instances occur fairly often.* We have tried to capture this idea in the notion of *uniform hardness.* Here we ask that not only is the language not feasibly computable but it has no long easy intervals. From the point of view that SAT is like this, it is very natural to study such languages.

Our final motivation is to attempt to relate some complexity-theoretical hypotheses to structural ones for appropriate reducibilities. Homer [15] has one example for the noncomputable languages under honest polynomial time reductions. Specifically, Homer proved that there if $P = NP$ then there exist minimal honest polynomial time degrees. Homer's languages are necessarily noncomputable, since Ladner's argument above demonstrates that there are no minimal honest polynomial time computable degrees. In our setting we get a nice consequence of the hypothesis that P = PSPACE. It seems that getting the consequence to fail is related not to punching holes in languages, but to the existence of certain types of one-way functions, although we do not explore this here.

2 Uniformly Hard Sets

We define "uniformly hard sets" to capture the notion of sets that do not have large gaps where they may be easily computable.

Definition 2.1 Let $m, n \in \mathbb{N}$. For $A \subseteq \Sigma^*$, we let $A \upharpoonright [m,n]$ denote

$$\{x \in A : m \leq |x| \leq n\}.$$

If $n = m$ we write $A \upharpoonright [n]$.

Definition 2.2 We say that $A \subseteq \Sigma^*$ is *uniformly hard* iff for any language $B \in$ P, there is a $k \in \mathbb{N}$ such that for all $m \geq 2$,

$$A \upharpoonright [m, m^k] \neq B \upharpoonright [m, m^k].$$

The reader should note that an equivalent definition would say that $A \upharpoonright [m, m^k] \neq B \upharpoonright [m, m^k]$ for *sufficiently large* m. Uniform hardness is not nearly as restrictive as almost-everywhere hardness as defined by Geske [13], where no polynomial-time algorithm can accurately characterize an infinite subset of the strings.

Definition 2.3 A set A is *polynomial-time many-one honest reducible* to B if there exists a polynomial-time computable function f and a polynomial p such that

1. $f : \Sigma^* \to \Sigma^* \cup \{T, F\}$
2. If $f(x) \in \Sigma^*$ then $p(|f(x)|) \geq |x|$ (honesty)
3. $x \in A$ if and only if $f(x) \in B \cup \{T\}$.

We require honesty so we can find inverses to f using our P = PSPACE assumption. We require T and F because in Ladner's proof one gets long stretches of nothingness and we would like an honest reduction to have something to map a positive instance to.

Uniformly hard sets are upward closed under these reductions.

Lemma 2.1 *If A is uniformly hard and polynomial-time many-one honest reducible to B then B is also uniformly hard.*

The polynomial-time many-one honest reductions give a partial order to the uniformly hard sets. A "polynomial-time many-one honest degree" is an equivalence class under this ordering.

One might expect to prove that if SAT is not uniformly hard we can derive some interesting collapse. But the possibility remains that at those inputs where SAT is hard, it could be very hard. This possibility is exactly captured by generic oracles.

Theorem 2.2 *Relative to any generic oracle, SAT and every other language in PSPACE is not uniformly hard yet the polynomial-time hierarchy is infinite.*

Proof. Let M be a relativized Turing machine such that for any A, M^A accepts a PSPACE^A-complete set L^A. We can assume without loss of generality that $M^A(x)$ only queries strings in A of length less than $|x|$.

By Lemma 2.1 we need only show that L^G is not uniformly hard relative to generic oracles G. Since $M^A(x)$ cannot query whether x is in A, the generic conditions ensure that

$$L^G \upharpoonright [n, 2^n] = G \upharpoonright [n, 2^n]$$

for infinitely many n. Since G is in P^G, we have that L^G is not uniformly hard relative to G.

Since Yao's proof [27] that the polynomial-time hierarchy is infinite relative to some oracle is a finite extension argument, the polynomial-time hierarchy is infinite relative to all generics (see also [9]). □

Uniformly hardness also has an important role in the area of "hardness versus randomness" [22, 5, 16]. One way to accurately state the recent result of Impagliazzo and Wigderson [16] is as follows.

Theorem 2.3 (Impagliazzo-Wigderson) *If a language in E cannot be accepted by any $2^{o(n)}$-size circuit family then every language in BPP is not uniformly hard.*

3 Main Result

Theorem 3.1 *If $P = PSPACE$ then there is a minimal uniformly hard polynomial-time many-one honest degree of a computable set.*

Proof. We must construct a uniformly hard set A so that for all $B <_m^h A$, B is not uniformly hard. To achieve this goal, we use an infinitary priority argument and satisfy the requirements below.

Let W_e denote the e-th P-time language so that "$x \in W_e$" is computable in time $|x|^e$.

In the following the variables n, m, k, s range over $\mathbb{N}$ and z, x, y over Σ^*. To make A uniformly hard we meet the requirements.

$$\mathcal{R}_e : (\forall^\infty\ n)[A \upharpoonright [n] \neq W_e \upharpoonright [n]].$$

Here, $\forall^\infty\ n$ denotes "for almost all n." Let φ_e denote the e-th poly-time honest m-reduction, so that for all x, $\varphi_e(x)$ is computable in time $|x|^e$, and

$$(|\varphi_e(x)|)^e > |x|.$$

To make A have minimal degree we must meet the requirements below.

$$\mathcal{M}_e : (A \leq_m^h \varphi_e^{-1}(A)) \vee (\varphi_e^{-1}(A) \text{ is not uniformly hard}).$$

Initially we make $A = \Sigma^*$ and every string "active." As we see below, as the construction progresses we will delete elements from A. As usual, in the construction specified below and later, languages are identified with their characteristic functions.

Meeting $\mathcal{R}_e$ in isolation. Meeting the $\mathcal{R}_e$ requirements is not too difficult. In isolation for each length $n > 2^e$, we pick an active x of length n and make $W_e(x) \neq A(x)$.

Meeting $\mathcal{M}_e$ in isolation. The construction will proceed in stages $s \in \mathbb{N}$. We will more or less at stage s decide the fate of all strings z with $|z| = s$. Associated with $\mathcal{M}_e$ will be a *current parameter* $k = k(e, s)$. We guarantee that $k(e, s) \geq k(e, s+1)$ for all e and s. It might be that $k(e, s) \to \infty$ or it might come to a limit. The value $k(e, s)$ goes to ∞ indicates that $\varphi_e^{-1}(A)$ is not uniformly hard.

Intuitive Overview The main idea is the following. We will try to divide the universe into sets of strings $[s_0, s_0), [s_1 = s_0^k, s_1^k), \ldots$, where $[s_i, s_i^k) = \{z : s_i \leq |z| < s_i^k\}$. We consider such half open intervals in turn. We wish to see *many* strings z with the image $\varphi_e(z)$ mapped to some length n in the interval $[s_i, s_i^k)$. Using the PSPACE=P assumption we can figure this out, and moreover, we can restrict diagonalization to only those strings which are images (or as we see "associates of images") of strings in the domain of φ_e. Then we will argue that we can, again using P=PSPACE, invert the φ_e map to make $A \leq_m^{h,P} \varphi_e(A)$. The fact that there are *many* such images, still allows us to make A uniformly hard.

One the other hand if there are not *many* then there are only *few* strings mapping into the relevant interval. In this case, we will not use any such image for diagonalization (that is, inactivate), and hence $\varphi_e(A)$ looks like $\varphi_e(\emptyset)$ for that interval, anyway. In that case we will reset make the

relevant k bigger, and argue that if this re-occurs infinitely often, there will be infinitely many easy intervals whose lengths dominate all polynomials, and hence $\varphi_e(A)$ cannot be uniformly hard.

Details Suppose that we have just reset k[1] and now have a particular value in mind. At some stage we will begin to look at $\mathcal{M}_e$ again. Suppose that s_0 is such a stage. We choose s_0 large enough such that $s_0^k < 2^{\sqrt{s_0}}$ and $s_0 > 2^e$.

Now we will act for $\mathcal{M}_e$ at stage s_0^k, and process those strings with lengths n where $s_0 \leq n < s_0^k$. Consider those strings x with $s_0 \leq |x| < s_0^k$. Let t_m be the number of active strings of length m, at the stage that $\mathcal{M}_e$ asserts control, and $t = \min_{s_0 \leq m < s_0^k} t_m$. At the end of the stage we will guarantee that there are at least $w = t/(s_0^k + 1)$ active strings remaining at each of these lengths.

We see if there is any length n with $s_0 \leq n < s_0^k$ such that

• there exist at least $t/(1 + (1/s_0^k))$ active strings x with $|x| = n$, and such that there exist a z with $\varphi(z) = x$.

There are two cases depending on whether the answer to • is *yes* or *no*.

Case 1. *The answer to • is yes*

For each $m \neq n$ with $s_0 \leq m < s_0^k$, delete all but the lexicographically least w many strings and make these deleted strings inactive. Let $B(n, s)$ denote the remaining strings of length n.

Now in order of $m \neq n$, and in lexicographic order of $x \in B(m, s)$, define the *e-associate* $a_e(x)$ to be lexicographically least active string of length n. Then make $a_e(x)$ inactive. That leaves at least $2w$ strings of length n. Associate in the same way the last w strings of length n with the first w and make the first w inactive. Delete and make inactive all remaining strings of length n.

After this stage, all strings which have not been removed from A will either be active or of the form $a_e(x)$ for some active x; that is associated with some active x. We will ensure that, henceforth, any active x and its associate $a_e(x)$ will enter or leave A *together*. That is, our promise is the following. For every active x,

$$x \in A \text{ iff } a_e(x) \in A.$$

Now we reset s_1 (the new s_0) to be s_0^k, and at stage s_1 repeat the above with s_1 in place of s_0. If the answer remains *yes* then we use $s_2 = s_1^k$, etc.

Outcome k. Suppose then that the answer is *always yes*[2]. In that case, we claim that $A \leq_m^h \varphi_e^{-1}(A)$. To see this, to decide if $z \in A$ from the $\varphi_e^{-1}(A)$ oracle, if $|z| < s_0$ then use a table lookup. Otherwise, compute i such that $s_i \leq |z| \leq s_i^k$. Using the assumption P = PSPACE, we can determine the active strings and their associates as follows.

For a *single* e Initially all strings y with $s_i \leq |y| \leq s_i^k$ are active. Hence for a single e, we merely need to see if there is a length n with $s_i \leq n < s_i^k$ for which there are at least $2^{s_i}(1 + (s_i^k)^{-1})$ many

[1] That is, we have made $k(e, s)$ into a different value overriding, and bigger than, all previous $k(e, s')$ for $s' < s$.

[2] As we see in the next case, if the answer is ever "no" then we will reset k. Hence the only two possibilities are "always yes" (for *some* k) or infinitely often "no."

x with $|x| = n$ and $\varphi_e(q) = x$ for some q. Determination of whether this is true involves counting strings of length n, for each n in the relevant range, determining if there is a q with $\varphi_e(q) = x$. But since φ_e is honest, a PSPACE oracle can determine whether there is such a q for each x and using the PSPACE oracle, again, can determine the number of such x for each n[3]. Since the answer for this k is always yes, there will be some least such n. then the assignation of associates from the other $B(m, s)$ involves lexicographic cycling through each $B(m, s)$ and generating lexicographically the members of $B(n, s)$. Again this can be done with $PSPACE$.

Now if z is not active or an associate then $z \notin A$. If z is an associate then $z \in A$ iff $\varphi_e^{-1}(z) \in \varphi_e^{-1}(A)$, where $\varphi_e^{-1}(z)$ is any element with $\varphi_e(\varphi_e^{-1}(z)) = z$ found using a PSPACE oracle. If z is active then $z \in A$ iff $\varphi_e^{-1}(a(z)) \in \varphi_e^{-1}(A)$. We call this process the outcome k.

Case 2. *The answer to* $\bullet$ *is no.* In this case our action is to delete all active strings z with $s_0 \leq |z| \leq s_0^k$ such that $\exists x[\varphi_e(x) = z]$. This leaves at least w active strings at each length.

We reset $k(e, s + 1) > k(e, s)$. We will later attend $\mathcal{M}_e$, again repeating these steps with a suitably longer s_0.

The outcome ∞. Suppose that this *no* case repeats itself infinitely often. The we claim that $\varphi_e^{-1}(A)$ is not uniformly hard. Let k be given. Compute a k_i and s_i sufficiently large that

$$(s_i + 1)^{eke} < s_i^{k_i},$$

and for which the outcome is *no*. Consider the interval of lengths $[(s_i + 1)^e, (s_i + 1)^{ek}]$. We claim that

$$\varphi_e^{-1}(\emptyset) \upharpoonright [(s_i + 1)^e, (s_i + 1)^{ek}] =$$
$$\varphi_e^{-1}(A) \upharpoonright [(s_i + 1)^e, (s_i + 1)^{ek}].$$

The point is that by the $|x|^e$-honesty of φ_e, it can only be that

$$\varphi_e^{-1}([(s_i + 1)^e, (s_i + 1)^{ek}]) \subseteq$$

$$\Sigma^* \upharpoonright [s_i, (s_i + 1)^{eke}] \subseteq \Sigma^*[s_i, s_i^{k_i}].$$

But since we make sure that there are no z in $[s_i, s_i^{k_i}]$ with z in the range of φ_e, we get the claim.

Therefore, for each k there is a m with $\varphi_e^{-1}(A) \upharpoonright [m, m^k] = \varphi_e^{-1}(\emptyset) \upharpoonright [m, m^k]$. Hence, $\varphi_e^{-1}(A)$ is not uniformly hard.

In summary In summary either we get the outcome k for some $k \in \mathbb{N}$, in which case $A \leq_m^h \varphi_e^{-1}(A)$, or $k(e, s) \to \infty$, the so-called Π_2 outcome which witnesses the fact that $\varphi_e^{-1}(A)$ is not uniformly hard.

Combining Requirements.

The construction will proceed in stages. The actual action is determined by recursion. The requirements are processed in some priority ordering, which, initially will be $\mathcal{R}_0, \mathcal{M}_0, \mathcal{R}_1, \mathcal{M}_1, \ldots$. As we will see this will change in the construction, at least for the $\mathcal{M}_j$. Any $\mathcal{R}_e$ requirement, will have at most $e - 1$ $\mathcal{M}_j$ requirements of higher priority. When they stop processing strings

[3]Probably, one could do all of this with only a $\#P$ oracle.

(by inactivating and deleting), then $\mathcal{R}_e$ can choose the least unused string of length m left to diagonalize. So it is simply a matter of ensuring that there are enough length m strings left. Thus, basically there is no problem with any number of $\mathcal{R}_j$ type requirements combining with a single $\mathcal{M}_e$. This is because we retain at least at $2^{-\sqrt{m}}$ fraction of the active strings every time length m is processed. We only bring in the $\mathcal{R}_e$ requirements slowly. We ensure that length m is processed for at most $\log\log m$ requirements.

The problems become more subtle when we consider more than one $\mathcal{M}_e$ type requirements, say $\mathcal{M}_e$ and $\mathcal{M}_f$ with $e < f$. (Two requirements are representative, and the inductive strategies continue in the obvious way.) First we will always ensure compatibility of the $k(e,s)$ and $k(f,s)$ by ensuring that at each stage s they are towers of 2, and hence either $k(e,s)$ is a root of $k(f,s)$ or vice versa. We will ask that they be distinct. We make the initial value of $k(i,0)$ a tower of 2's of height $2i+1$.

Now what turns out to be important is the relative sizes of $k_e = k(e,s)$ and $k_f = k(f,s)$ rather than their priorities, since we will use a dynamic priority ordering, in a sense to be described below. To begin with suppose that we have $k_e < k_f$. Then we will have $k_f = k_e^d$ for some d. Both $\mathcal{M}_e$ and $\mathcal{M}_f$ will deal with the same s_0 in the sense of the isolated strategy. But, assuming that $\mathcal{M}_e$ has the k_e outcome each time, the f-interval $[s_0, s_0^{k_f}]$ will consist of d e-intervals of the form

$$[s_0, s_0^{k_e}], [s_1 = s_0^{k_e}, s_1^{k_e}], ..., [s_d, s_d^{k_e}].$$

Call these intervals $I_1, ..., I_d$.

Now we regard the output of the basic e-module applied to I_t as producing equivalence classes of strings with two strings, one active and one its associate. Consider the active string the representative of this class.

In future applications of the basic modules for the $\mathcal{R}$ and $\mathcal{M}$ requirements, treat any query to an element of an equivalence class as a query to the active representative. If the representative is deleted then so should all of the other elements in the class. This may in turn yield larger equivalence classes with still one active representative.

Thus, at the very stage $t = s_d^{k_e} = s_0^{k_f}$ we get to process for $\mathcal{M}_f$, all of the $I_1, ..., I_d$ will have been e-processed, and many strings will have been inactivated, deleted, or associated. $\mathcal{M}_f$ simply deals with this smaller universe, seeing each pair as a single element.

To complete the description of the construction, we remark that if the answer to $\bullet$ is no, we will reset $k(e,s)$ (or $k(f,s)$ as the case may be) to be the next largest unused tower of 2. We also reset s_0 to start at the next appropriate point for it. For instance, if $k(e,s)$ was reset while processing $[s_j, s_j^{k_e})$, because the answer to $\bullet$ was no then we would reset s_0 for it to be $s_0^{k_f}$. For convenience, the tower of 2' we reset k_e to be determines its new priority which means when it gets processed. Thus if the new k_e is k' then we will make $\mathcal{M}_e$ now have priority below $\mathcal{R}_{k'}$. Note that if the $k(e,s)$ has a limit then this causes no problem. There is a fixed depth of higher priority requirements that get processed before it and hence a PSPACE oracle can determine the active elements, equivalence classes etc.

If the $k(e,s)$ has no limit, then the result is infinitely often the same as having $\emptyset$ as an oracle.

There are no problems extending this to arbitrary depths of requirements, and the calculation of the exact numbers merely obscures the proof. □

4 Future Directions

We show that Ladner's nonminimality result appears to require creating large gaps of easiness. There are several directions one can take this research including

- Weaken or eliminate the P = PSPACE assumption.
- Find a complexity theoretical hypothesis (such as a strong one way hypothesis) for which there are no languages minimal in our sense.
- Look at density issues among the uniformly hard sets. First one needs a good definition as to when one set is "uniformly harder" than another.
- Look at decidability in the uniformly hard degrees.

References

[1] K. Ambos-Spies, "On the structure of polynomial time degrees" in STACS 84, Lecture Notes in Computer Science, Vol. 166 (1984) 198-208, Springer-Verlag.

[2] K. Ambos-Spies, "On the structure of the polynomial time degrees of recursive sets," Technical report 206 (1985), Abteilung Informatik, Universität Dortmund.

[3] K. Ambos-Spies, "Minimal pairs for polynomial time reducibilities," in *Computation and Proof Theory*, Lecture Notes in Computer Science, Vol. 270 (1987) 1-13.

[4] K. Ambos-Spies and A. Nies, "The Theory of The Polynomial Time Many-One Degrees is Undecidable," STACS 1992, 209-218.

[5] L. Babai, L. Fortnow, N. Nisan, and A. Wigderson. BPP has subexponential simulations unless EXPTIME has publishable proofs. *Computational Complexity*, 3:307–318, 1993.

[6] J. Balcázar and J. Diaz, "A note on a theorem of Ladner," Information Processing Letters, Vol. 15 (1982), 84-86.

[7] J. Balcázar, J. Diaz, and J. Gabarro, *Structural Complexity* Vol. 1 and 2 (1987,1989), Springer.

[8] L. Berman and J. Hartmanis, *On isomorphism and density of NP and other complete sets*, SIAM Journal on Computing, Vol. 1 (1977), 305-322.

[9] M. Blum and R. Impagliazzo. Generic oracles and oracle classes. In *Proceedings of the 28th IEEE Symposium on Foundations of Computer Science*, pages 118–126. IEEE, New York, 1987.

[10] P. Chew and M. Machtey, "A note on structure and looking back applied to the relative complexity of computable sets," *J. Comput. Sys. Sci.*, Vol. 22 (1981), 53-59.

[11] R. Downey, "Nondiamond theorems for polynomial time reducibility," *J. Comput. Sys. Sci.,* Vol. 45 (1992) 385-395

[12] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, (1979), Freeman.

[13] J. Geske. *On the structure of intractable sets*. PhD thesis, Iowa State University, Ames, Iowa, 1987.

[14] S. Homer, "Some properties of the lattice of NP sets," in *Workshop on Recursion Theoretical Aspects of Computer Science,* Purdue University, (1981) 18-22.

[15] S. Homer, "Minimal degrees for polynomial reducibilities," *Journal of the ACM*, Vol. 34 (1987), 480-491.

[16] R. Impagliazzo and A. Wigderson. P = BPP if E requires exponential circuits: Derandomizing the XOR lemma. In *Proceedings of the 29th ACM Symposium on the Theory of Computing*. ACM, New York, 1997, 220-229.

[17] R. Karp, "Reducibility among combinatorial problems," in *Complexity of Computer Computations,* (R. Miller and J. Thatcher, eds) (1972), 85-104, Plenum Press.

[18] R. Ladner, "On the structure of polynomial-time reducibility" *J. Assoc. Comput. Mach.,* Vol. 22 (1975), 155-171.

[19] R. Ladner, N. Lynch and A. Selman, "A comparison of polynomial time reducibilities," *Theoretical Computer Science*, Vol. 1 (1975), 103-123.

[20] L. Landweber, R. Lipton and E. Robertson, "On the structure of NP and other complexity classes," *Theoretical Computer Science*, Vol. 15 (1981), 181-200.

[21] K. Melhorn, "Polynomial and Abstract Subrecursive Classes," *J. Comput. Sys. Sci.* 12 (1976) 147-178.

[22] N. Nisan and A. Wigderson. Hardness vs. randomness. *Journal of Computer and System Sciences*, 49:149–167, 1994.

[23] K. Regan, "On diagonalization methods and the structure of language classes," in *Fundamentals of Computation Theory,* (M. Karpinski, ed.) Springer Verlag Lecture Notes in Computer Science, Vol. 158 (1983), 368-380.

[24] U. Schöning, "A uniform approach to obtain diagonal sets in complexity classes," *Theoretical Computer Science,* Vol. 18 (1982), 95-103.

[25] J. Shinoda and T. Slaman, "On the Theory of $PTIME$ Degrees of Recursive Sets, *J. Comput. Sys. Sci.* Vol. 41 (1990), 321-366.

[26] R. A. Shore and T. A. Slaman, "The p-T degrees of the recursive sets: lattice embeddings, extensions of embeddings and the two-quantifier theory," *Theoretical Computer Science*, 97, (1992), 763-784.

[27] A. Yao. Separating the polynomial-time hierarchy by oracles. In *Proceedings of the 26th IEEE Symposium on Foundations of Computer Science*, pages 1–10. IEEE, New York, 1985.

A Appendix

For completeness and to aid discussion we give two proofs that if P $\neq$ NP then there exists an incomplete set A in NP $-$ P.

Both proofs have a similar set up. First we assume that P $\neq$ NP. Every NP-complete language is not in P with this assumption and we will focus on one of them, namely SAT, the language of satisfiable Boolean formula.

We have two sets of requirements to fulfill. Let $M_1, M_2, \ldots$ be an enumeration of deterministic Turing machines clocked so that the machine $M_i(x)$ runs in time $|x|^i$ and captures all of the languages in P. We also have a similar list f_i of the polynomial-time computable functions.

1. R_i: $A \neq L(M_i)$.
2. S_i: For some x, $x \in$ SAT and $f_i(x) \notin A$ or $x \notin$ SAT and $f_i(x) \in A$.

In addition we need to guarantee that A is in NP.

A.1 Proof by blowing holes in SAT

Our set A will be defined using a function f by

$$A = \{x \mid x \in \text{ SAT and } f(|x|) \text{ is even}\}.$$

Note that if we make $f(n)$ computable in polynomial in n time then A will be in NP.

The function f will be set to the current stage of the construction. Intuitively in stage $2i$, we keep $f(n) = 2i$ for large enough n until condition R_i is fulfilled. If R_i is never fulfilled then the set A will be equal to $L(M_i)$ and a finite difference from SAT condradicting the assumption that P $\neq$ NP.

In stage $2i+1$ we keep $f(n) = 2i+1$ until condition S_i is fulfilled. If S_i is never fulfilled then A will be finite and SAT reduces to A via f_i which would put SAT in P, again contradicting the fact that P $\neq$ NP.

The trick is to do this while keeping f polynomial-time computable. We do this by *delayed diagonalization*, i.e., we do not start a new stage until we see the requirement for the previous stage has been fulfilled on inputs so small we can test it. Thus we do not start a new stage until well after the old requirements are fulfilled.

We now formally define $f(n)$ inductively in n. Let $f(0) = f(1) = 2$. For $n \geq 1$ we define $f(n+1)$ as follows: If $\log^{f(n)} n \geq n$ then let $f(n+1) = f(n)$. Otherwise we have two cases:

$f(n) = 2i$: Check to see if there is an input x, $|x| \leq \log n$ such that either

1. $M_i(x)$ accepts and either $f(|x|)$ is odd or x is not in SAT, or

2. $M_i(x)$ rejects and $f(|x|)$ is even and x is in SAT.

If such an x exists then let $f(n+1) = f(n)+1$ otherwise we let $f(n+1) = f(n)$.

$f(n) = 2i+1$: Check to see if there is an input x, $|x| \leq \log n$ such that either

1. x is in SAT and either $f(|f_i(x)|)$ is odd or $f_i(x)$ is not in SAT, or
2. x is not in SAT and $f(|f_i(x)|)$ is even and $f_i(x)$ is in SAT.

If such an x exists then let $f(n+1) = f(n)+1$ otherwise we let $f(n+1) = f(n)$.

Since to compute $f(n)$ we only examine x with $|x| \leq \log n$ and

$$|x|^i \leq \log^i n \leq \log^{f(n)} n < f(n),$$

we can compute $f(n)$ in time polynomial in n. It is straightforward to check that $f(n)$ does not increase until the corresponding reqiurements if fulfilled and that if $f(n)$ remains constant for all large n then we will have violated the P $\neq$ NP assumption.

A.2 Proof by Padding SAT

Here the idea is to encode SAT questions of length n on inputs of length $f(n)$. Define the language L as

$$L = \{\phi 01^{f(n)-|n|-1} \mid \phi \text{ in SAT, and } |\phi| = n\}.$$

We will create a polynomial-time computable in n function f large enough so that L is not NP-complete but not so large as to make L in P.

We will keep $f(n) = n^i$ long enough to fulfill R_i and then let $f(n) = n^{i+1}$.

We define formally define an algorithm for computing $f(n)$. Let $i = 1$ initially. For each n in order we do the following: Let $f(n) = n^i$. Check to see if there is an input x, $|x| \leq \log n$ such that either

1. $M_i(x)$ accepts and x is not in L, or
2. $M_i(x)$ rejects and x is in L.

If so let $i = i+1$ otherwise leave i unchanged. Go onto the next n.

Since we are only checking very small x, we can compute f in polynomial time in n.

Suppose that L is in P. We then have that $L = L(M_i)$ for some i so $f(n) = n^i$ for suitably large n. But then we have an easy reduction from SAT to L and SAT would also be in P, violating our assumption.

So we have fulfilled all of the R_i requirements and i goes to infinity. Suppose some requirement S_j is not fulfilled. We then have a function f_j that reduces SAT to L. We want to show that we can now compute whether ϕ is in SAT in polynomial time.

Since f_j runs in time bounded by n^j we have that for all ϕ, $|f_j(\phi)| \leq |\phi|^j$. There must be some n_0 such that for all $n \geq n_0$, $f(n) = n^k$ for some $k > j$. We hardwire satisfiability for all inputs of length up to n_0.

Suppose we have a formula ϕ with $|\phi| > n_0$. If $f_j(\phi)$ is not in the range of f then $f_j(\phi)$ is not in L so ϕ is not in SAT. Otherwise, $f_j(\phi) = \psi 01^{f(m)-m-1}$ where $m = |\psi|$ and ϕ in SAT if and only ψ is in SAT. We have $f(m) = |f_j(\phi)| \leq |\phi|^j$ so $|\psi| = m \leq |\phi|^{j/k}$ if $f(m) = m^k$. Since $|\phi| > n_0$ we have $k > j$ so $|\psi| < |\phi|$. If $|\psi| \leq n_0$ then we know whether ψ and thus ϕ is in SAT. Otherwise we apply this algorithm recursively to ψ. Since $|\psi|$ gets smaller each step the algorithm runs in polynomial time.