On Sets Bounded Truth-Table Reducible to P-selective Sets *

Thomas Thierauf[†] Seinosuke Toda[‡] Osamu Watanabe[§]

July 26, 1996

Abstract

We show that if every NP set is polynomial-time bounded truth-table reducible to some P-selective set, then NP is contained in $\mathrm{DTIME}(2^{n^{O(1/\sqrt{\log n})}})$. In the proof, we implement a recursive procedure that reduces the number of nondeterministic steps of a given nondeterministic computation.

1 Introduction

The class NP is commonly considered as a class of problems that cannot be solved efficiently, that is, by polynomial-time bounded, deterministic Turing machines. Changing from (uniform) Turing machines to (nonuniform) circuits, one of the important questions in computational complexity theory is whether every NP problem is solvable by small, that is, polynomial-size, circuits. Furthermore, assuming that NP problems can indeed be solved by small circuits, it has been asked whether this is turn gives deterministic algorithms for NP faster than the known exponential ones. In other words, if NP is easy in the nonuniform complexity measure, how easy is NP in the uniform complexity measure? We study such type of questions in this paper. Karp and Lipton [KL82] have shown that if NP has small circuits then the Polynomial Hierarchy [Sto77] collapses, therby giving strong evidence that the assumption might not hold. Note however, that this does not answer the above question.

*Part of the work was done while the authors were visiting the University of Rochester, Department of Computer Science. This research is supported in part by JSPS/NSF International Collaboration Grant JSPS-ENGR-207/NSF-INT-9116781, DFG Postdoctorial Stipend Th 472/1-1, and NSF grant CCR-8957604.

†Abteilung Theoretische Informatik, Universität Ulm, 89068 Ulm, Germany. Email: thierauf@informatik.uni-ulm.de.

‡Department of Applied Mathematics, Nihon University, Setagaya-ku Sakura-Jyosui 3-25-40, Tokyo 156, Japan. Email: toda@math.chs.nihon-u.ac.jp.

§Department of Computer Science, Tokyo Institute of Technology, Tokyo 152, Japan. Email: watanabe@cs.titech.ac.jp.

Small circuits can be coded by *sparse sets* and vice versa. Therefore, the class of sets that have polynomial-size circuits coincides with the class of sets that are (polynomial-time) Turing reducible to some sparse set. We denote the latter class by $\mathrm{R}^{\mathrm{P}}_{\mathrm{T}}(\mathrm{SPARSE})$. Hence, the above question is equivalent to the following one: For which uniform deterministic complexity class $\mathcal{C}$ do we have $\mathrm{NP} \subseteq \mathrm{R}^{\mathrm{P}}_{\mathrm{T}}(\mathrm{SPARSE}) \Longrightarrow \mathrm{NP} \subseteq \mathcal{C}$?

Nontrivial answers to this question are only known for even stronger assumptions such as that NP is contained in certain subclasses of $\mathrm{R}^{\mathrm{P}}_{\mathrm{T}}(\mathrm{SPARSE})$. For example, Mahaney [Mah82] showed that if every NP set is many-one reducible to some sparse set then $\mathrm{P} = \mathrm{NP}$. That is,

$$\mathrm{NP} \subseteq \mathrm{R}^{\mathrm{P}}_{\mathrm{m}}(\mathrm{SPARSE}) \Longrightarrow \mathrm{NP} = \mathrm{P}.$$

Ogiwara and Watanabe [OW91] extended Mahaney's result to bounded truth-table reductions, that is,

$$\mathrm{NP} \subseteq \mathrm{R}^{\mathrm{P}}_{\mathrm{btt}}(\mathrm{SPARSE}) \Longrightarrow \mathrm{NP} = \mathrm{P}.$$

This result has been improved further more recently, see [AHH$^+$93]. However, it is open whether the result can be improved to $b(n)$-bounded truth-table reducibility for some nonconstant function $b(n)$. Indeed, Saluja [Sal93] showed that, at least with the technique used by Ogiwara and Watanabe, such an improvement is impossible. Furthermore, for $b(n) = \omega(\log n)$, Homer and Longpré [HL94] (see also [AHH$^+$93]) constructed an oracle relative to which $\mathrm{NP} \subseteq \mathrm{R}^{\mathrm{P}}_{b(n)\text{-tt}}(\mathrm{SPARSE})$, but P is different from NP.

Small circuits can also be coded as *leftcuts of real numbers* and vice versa [Ko83, Sel82b]. Leftcuts can be formalized in terms of P-selective sets [Sel82b]. Therefore, the class of sets that have polynomial-size circuits coincides with the class of sets that are (polynomial-time) Turing reducible to some P-selective set. Let SELECT denote the class of P-selective sets. Thus, we have $\mathrm{R}^{\mathrm{P}}_{\mathrm{T}}(\mathrm{SELECT}) = \mathrm{R}^{\mathrm{P}}_{\mathrm{T}}(\mathrm{SPARSE})$. However, for reductions that are more restrictive than the Turing reduction, classes obtained by reducing to P-selective sets can be different from classes obtained by reducing to sparse sets. For example, Watanabe [Wat90] showed $\mathrm{R}^{\mathrm{P}}_{\mathrm{tt}}(\mathrm{SELECT}) \neq \mathrm{R}^{\mathrm{P}}_{\mathrm{tt}}(\mathrm{SPARSE})$ (see [HHO$^+$93] for more separations). Hence, it is interesting to investigate the consequences of NP being reducible to P-selective sets with respect to some more restrictive type of reducibility.

Selman [Sel79] showed that if every NP set is many-one reducible to some P-selective set then $\mathrm{P} = \mathrm{NP}$. Assuming that NP sets are (unbounded) truth-table reducible to P-selective sets, Toda [Tod91] and Beigel [Bei88] showed that NP problems can be solved efficiently by randomized Las Vegas type algorithms, a class denoted by R.

$$\mathrm{NP} \subseteq \mathrm{R}^{\mathrm{P}}_{\mathrm{tt}}(\mathrm{SELECT}) \Longrightarrow \mathrm{NP} = \mathrm{R}. \tag{1}$$

In this paper, we show a deterministic upper bound on NP when considering bounded truth-table reductions. Namely, we show

$$\mathrm{NP} \subseteq \mathrm{R}^{\mathrm{P}}_{\mathrm{btt}}(\mathrm{SELECT}) \implies \mathrm{NP} \subseteq \mathrm{DTIME}(2^{n^{O(1/\sqrt{\log n})}}). \tag{2}$$

Let us give a brief outline of our proof. We start by sketching the idea to prove equation (1). The assumption $\mathrm{NP} \subseteq \mathrm{R}^{\mathrm{P}}_{\mathrm{tt}}(\mathrm{SELECT})$ is essentially used to show:

(∗) for a given polynomial-time nondeterministic Turing machine M and a string x, if M on input x has *exactly one* accepting path, then the path is computable in deterministic polynomial time.

For M and x as above, the nondeterministic computation of M on x can be viewed as a (binary) tree T. Using the randomized hashing technique of Valiant and Vazirani [VV86], one can construct subtrees $T_1, \ldots, T_m$ of T, all having the same root as T, such that if T has an accepting path then, say, $m/4$ of $T_1, \ldots, T_m$ have *exactly one* accepting path. Then from property (∗), for the T_k's having exactly one accepting path, one can compute this path. Thus, by choosing T_k randomly for several times, one can compute some accepting path of T with high probability if there are any. This is the idea of showing NP = R.

We also use (∗) for proving equation (2). Consider again a nondeterministic computation tree T as above. Using our stronger assumption, namely that $\mathrm{NP} \subseteq \mathrm{R}^{\mathrm{P}}_{\mathrm{btt}}(\mathrm{SELECT})$, we can construct subtrees $T'_1, \ldots, T'_n$ of T such that if T has some accepting path, then some T'_k has exactly one accepting path, and, by property (∗), such a path can be computed in polynomial time. The important point here is that the number of subtrees, n, can be chosen fairly small compared with m from above, or with the number of paths in T. Hence, the original NP question "Does T have an accepting path?" is reduced to another NP question "Is there a k such that T'_k has an accepting path?", and in addition, the size of the search space in the latter NP question (searching for some k) is much smaller than in the former one (searching for some path). Hence, for solving the reduced NP question, one needs a smaller number of nondeterministic guesses. We show how to apply this process *recursively*, thereby successively decreasing the search space of the reduced NP questions obtained. In total, this yields a subexponential algorithm to solve the original problem *deterministically*.

Related work was done by Jenner and Torán [JT95]. They showed under the assumption that functions that can be computed in polynomial time by making truth-table queries to NP can already be computed in polynomial time by making logarithmically many (adaptive) queries to NP (in symbols, $\mathrm{FP}^{\mathrm{NP}}_{\mathrm{tt}} = \mathrm{FP}^{\mathrm{NP}[\log]}$), it follows that $\mathrm{NP} \subseteq \mathrm{DTIME}(2^{n/\log^k n})$, for any $k \geq 1$. Note that their assumption is seemingly weaker than ours since it is not hard to see that $\mathrm{NP} \subseteq \mathrm{R}^{\mathrm{P}}_{\mathrm{tt}}(\mathrm{SELECT})$ implies $\mathrm{FP}^{\mathrm{NP}}_{\mathrm{tt}} = \mathrm{FP}^{\mathrm{NP}[\log]}$, but the converse implication is not known to hold. It seems, however, not possible to obtain our stronger upper bound on NP from their assumption by their technique [Tor93].

Most notably, we mention that our result has been improved recently. Namely, Agrawal and Arvind [AA94], Beigel, Kummer, and Stephan [BKS95], and Ogihara [O94] showed that $\mathrm{NP} \subseteq \mathrm{R}^{\mathrm{P}}_{\mathrm{btt}}(\mathrm{SELECT}) \implies \mathrm{NP} = \mathrm{P}$. In fact, the result holds up to quasi-linear truth-table reducibility, i.e., $O(n^{1-\epsilon})$, for any $\epsilon > 0$, [AA94, O94]. The principal method in all three papers is a standard search and pruning technique with the goal to find a satisfying assignment for a given Boolean formula F in polynomial time (if there exist any). During the search, a set X of subformulas of F is maintained such that the following invariant is fulfilled: $F \in SAT \iff X \cap SAT \neq \emptyset$. Initially, $X = \{F\}$. While going breadth-first through the self-reduction tree of F, X is successively extended and then pruned again such that the size of X remains polynomially bounded. The pruning task is to determine an $x \in X$ such that if $X \cap SAT \neq \emptyset$ then $(X - \{x\}) \cap SAT \neq \emptyset$. Then x can be pruned from X since there will still be a satisfiable formula in X if there are any, thereby maintaining the invariant. By assumption, formulas in X can be reduced to a P-selective set. The crucial point in their proofs is to also produce new Boolean formulas by or-ing together some (apppropriate) formulas of X, and to reduce them to the P-selective set as well. Since the new instances are related to the formulas in X (by the or-function), this must be reflected in the way these strings are mapped by the reduction. Exactly this property is used to find an instance x to prune as described above.

Thus, our approach is completely different from the one's mentioned above. Roughly speaking, the proofs in [AA94, BKS95, O94] essentially use the fact that there are NP complete sets that are (disjunctively) self-reducible and have an or-function in order to make their searching technique work. In contrast, we use the completeness of certain NP sets, but we don't use such or-functions, and thus, we need to establish a more elaborate searching technique. Therefore, although the main result we will derive in this paper is already subsumed, we think that our proof technique is interesting for its own, and hence, we encourage the reader to continue reading!

2 Preliminaries

We follow the standard definitions and notations in computational complexity theory (see, e.g., [BDG88, BDG91]).

We fix an alphabet $\Sigma = \{0, 1\}$. For any set $X \subseteq \Sigma^*$, we denote the complement of X as $\overline{X} = \Sigma^* - X$. Natural numbers are encoded in Σ^* by using their binary representation. For any string x, let $|x|$ denote the length of x, and for any set X, let $||\, X \,||$ denote the cardinality of X. We consider a standard one-to-one pairing function from $\Sigma^* \times \Sigma^*$ to Σ^* that is computable and invertible in polynomial time. For strings x and y, we denote the output of the pairing function by (x, y); this notation is extended to denote tuples. For example (x, y, z)

is defined as $((x, y), z)$. For a function f, we simply write $f(x, y)$ instead of $f((x, y))$.

We use the standard Turing machine as our computation model. P (resp., NP) denotes the class of languages that can be recognized by some polynomial-time deterministic (resp., nondeterministic) Turing machine. For a nondeterministic Turing machine M, we assume that every nondeterministic configuration of M has at most two succeeding ones. Hence, each nondeterministic computation of M on a given input can be described by a string w, where the ith bit of w indicates which branch to take at the ith nondeterministic branch point. In this context, we call a string w a *path of* M, and, in case that w leads to an accepting configuration of M on a given input, we call w an *accepting path of* M *on that input*.

For any sets A and B, we say that A is *many-one reducible to* B (and write $A \leq_{\mathrm{m}}^{\mathrm{P}} B$) if there is some polynomial-time computable function f, the *reduction*, such that for any $x \in \Sigma^*$, we have $x \in A \iff f(x) \in B$. A set C is called *NP-complete* if (i) every NP set is many-one reducible to C, and (ii) C itself is in NP. The reducibility notions we are interested in are generalization of the many-one reduction. We say that A is *truth-table reducible to* B (and write $A \leq_{\mathrm{tt}}^{\mathrm{P}} B$) if there are two polynomial-time computable functions, *generator* g that, for a given $x \in \Sigma^*$, produces a set of strings, and *evaluator* e that, when knowing which of the strings produced by g are in B, decides membership of x in A. That is, for any $x \in \Sigma^*$,

$$x \in A \iff e(x, g(x), g(x) \cap B) = 1,$$

where we assume that $g(x)$ (resp., $g(x) \cap B$) is encoded as a string. For any $b(n) \geq 0$, we say that A is *$b(n)$-truth-table reducible to* B (and write $A \leq_{b(n)\text{-tt}}^{\mathrm{P}} B$) if the generator g produces at most $b(n)$ strings for each input of length n. We say that A is *bounded-truth-table reducible to* B (and write $A \leq_{\mathrm{btt}}^{\mathrm{P}} B$) if A is $\leq_{k\text{-tt}}^{\mathrm{P}}$-reducible to B, for some constant $k \geq 0$. *Hard* and *complete* sets with respect to these reducibilities are defined analogously as for the many-one reducibility.

For any class $\mathcal{C}$ of languages, let $\mathrm{R}_{\mathrm{T}}^{\mathrm{P}}(\mathcal{C})$, $\mathrm{R}_{\mathrm{tt}}^{\mathrm{P}}(\mathcal{C})$, $\mathrm{R}_{b(n)\text{-tt}}^{\mathrm{P}}(\mathcal{C})$, and $\mathrm{R}_{\mathrm{btt}}^{\mathrm{P}}(\mathcal{C})$ respectively denote the class of sets that are $\leq_{\mathrm{T}}^{\mathrm{P}}$-, $\leq_{\mathrm{tt}}^{\mathrm{P}}$-, $\leq_{b(n)\text{-tt}}^{\mathrm{P}}$, and $\leq_{\mathrm{btt}}^{\mathrm{P}}$-reducible to some set in $\mathcal{C}$.

P-selective sets were introduced by Selman [Sel79] as the polynomial-time analog of semi-recursive sets [Joc68]. A set A is *P-selective*, if there exists a polynomial-time computable function f, called a *P-selector for* A, such that for all $x, y \in \Sigma^*$,

1. $f(x, y) \in \{x, y\}$, and
2. if $x \in A$ or $y \in A$, then $f(x, y) \in A$.

Intuitively, f selects the one of the two given strings that is "more likely" to be in A. More formally, if $f(x,y)=x$ and $y \in A$, then $x \in A$. The class of P-selective sets is denoted as SELECT.

Ko [Ko83] showed that for every P-selective set A, using the P-selector function f of A, one can define a linear ordering on a quotient of Σ^* such that A is the union of an initial segment of this ordering. Toda [Tod91] modified this to an ordering on a given finite set Q (instead of Σ^*). Here, we use this ordering. That is, we define the relation $\preceq_{f,Q}$ on Q as follows. For all $x, y \in Q$,

$$x \preceq_{f,Q} y \iff \begin{array}{l} \text{there exist } z_1, \ldots, z_n \in Q \text{ such that} \\ f(z_i, z_{i+1}) = z_i \text{ for } i = 1, \ldots, n-1, \\ f(x, z_1) = x, \text{ and } f(z_n, y) = z_n. \end{array}$$

Define $x \cong_{f,Q} y \iff x \preceq_{f,Q} y \wedge y \preceq_{f,Q} x$. Then $\cong_{f,Q}$ is an equivalence relation on Q, and $\preceq_{f,Q}$ induces a linear ordering on the quotient $Q/\cong_{f,Q}$. This is reflected by the following partial ordering $\prec_{f,Q}$ on Q:

$$x \prec_{f,Q} y \iff x \preceq_{f,Q} y \ \wedge \ x \not\cong_{f,Q} y.$$

For simplicity, we omit the subscripts f and Q when both are clear from the context. For technical reasons, we introduce a minimum and a maximum element, denoted as $\perp$ and $\top$ respectively, such that $\perp \prec x \prec \top$, for all $x \in Q$.

It is easy to see that the relations $\prec$ and $\cong$ are decidable in polynomial time in $\sum_{x \in Q} |x|$. The crucial point is that $A \cap Q$ is an initial segment of Q with respect to $\preceq$. That is, we have

$$\exists\, z \in Q \cup \{\perp\} : \quad Q \cap A = \{\, y \in Q \mid y \preceq z \,\} \text{and } Q \cap \overline{A} = \{\, y \in Q \mid y \succ z \,\}.$$

We call a string z witnessing this property a *cutpoint* of A in Q (with respect to $\preceq$). A consequence of this property is that $\forall\, x, y \in Q$: $\ x \preceq y \wedge y \in A \implies x \in A$.

3 Main Result

In this section, we show that if all NP sets are bounded truth-table reducible to some P-selective set, then every NP set is solvable deterministically in $2^{n^{O(1/\sqrt{\log n})}}$ steps. We begin by recalling a result of Toda [Tod91] that will be used in our proof. We use a formulation in terms of promise problems.

Definition 3.1 [ESY84] *A* promise problem *is a pair of sets* (Q, R)*. A set* L *is called a* solution *of the promise problem* (Q, R)*, if for all* $x \in Q$*, we have* $x \in R \iff x \in L$*.*

In other words, if L is a solution of a promise problem (Q, R), then L coincides with R on all instances where the *promise* Q holds. That is, $Q \cap R = Q \cap L$.

Toda [Tod91] showed that if all NP sets are $\leq_{\mathrm{tt}}^{\mathrm{P}}$-reducible to some P-selective set, then the promise problem $(\mathit{1\text{-}SAT}, \mathit{SAT})$ has a solution in P, where $\mathit{1\text{-}SAT}$ is the set of Boolean formulas that have at most one satisfying assignment. We restate his theorem in a slightly more general form and include a proof for completeness.

Theorem 3.2 [Tod91] *If* $\mathrm{NP} \subseteq \mathrm{R}_{\mathrm{tt}}^{\mathrm{P}}(\mathrm{SELECT})$ *then, for any NP machine N the promise problem* $(1\text{-}L(N), L(N))$ *has a solution in P, where* $1\text{-}L(N)$ *is the set of strings x such that N has at most one accepting path on input x. Furthermore, if N is $p(n)$ time bounded, then the solution is in* $\mathrm{DTIME}(q_{\mathrm{T}} \circ p(n))$*, for some fixed polynomial q_{T}.*

Proof. Define the NP set *BitPATH* as follows. For a nondeterministic Turing machine N, a string x, $d, t \geq 1$, and $1 \leq i \leq d$,

$$(N, x, 0^d, 0^t, i) \in \mathit{BitPATH} \iff \begin{array}{l} \text{there exists } w \in \Sigma^{\leq d} \text{ such that} \\ N \text{ accepts } x \text{ on path } w \text{ in } t \text{ steps} \\ \text{and the } i\text{th bit of } w \text{ is } 0. \end{array}$$

By assumption, *BitPATH* is truth-table reducible to some P-selective set A. Let g be the generator and e the evaluator of the reduction, and let f be a P-selector for A.

Let N be an NP machine, and let polynomial p bound its running time. Consider an instance x, $|x| = n$, for N such that N has exactly one accepting path w on input x. Clearly, we can reconstruct w when knowing the answers to the questions "$z_i = (N, x, 0^{p(n)}, 0^{p(n)}, i) \in \mathit{BitPATH}$?", for $i = 1, \ldots, p(n)$.

Let Q be the set of strings queried to A on z_i by the generator of the truth-table reduction, for $i = 1, \ldots, p(n)$, i.e., $Q = \{\ y \mid y \in g(z_i), \text{ for some } i, 1 \leq i \leq p(n)\ \}$. If we know which point of Q is a cutpoint of A w.r.t. $\preceq_{f,Q}$, we would be able to get the correct answer to each query "$z_i \in \mathit{BitPATH}$?", thereby obtaining the unique accepting path w. Here, note that Q has only polynomially many elements; thus, we can try all elements y of Q and check whether we obtain an accepting path (namely, w) assuming that y is a cutpoint. (Note that we can easily verify whether a reconstructed path is an accepting path.) The algorithm in figure 1 makes this idea more precise. Here, N and p are fixed parameters.

Let M_N be a deterministic Turing machine that executes this algorithm. Clearly, $L(M_N)$ is a solution for $(1\text{-}L(N), L(N))$. Furthermore, there exists some polynomial q_{T} such that for any N, M_N halts in $O(q_{\mathrm{T}}(p(n)))$ steps. $\square$

Now, we prove our main theorem.

Theorem 3.3 *If* $\mathrm{NP} \subseteq \mathrm{R}_{\mathrm{btt}}^{\mathrm{P}}(\mathrm{SELECT})$ *then* $\mathrm{NP} \subseteq \mathrm{DTIME}(2^{n^{O(1/\sqrt{\log n})}})$.

UNIQUE-ACCEPTING-PATH$(x,\ |x| = n)$;

$Q \leftarrow \bigcup_{1 \le i \le p(n)} g(N, x, 0^{p(n)}, 0^{p(n)}, i)$;
for each $y \in Q \cup \{\perp\}$ **do**
  for $i \leftarrow 1$ **to** $p(n)$ **do**
    if the evaluator e accepts $(N, x, 0^{p(n)}, 0^{p(n)}, i)$ when the answers
      to $g(N, x, 0^{p(n)}, 0^{p(n)}, i)$ are given according to cutpoint y
    then $w_i \leftarrow 0$ **else** $w_i \leftarrow 1$;
    if $w = w_1 \cdots w_{p(n)}$ is an accepting path of N on input x
    then accept;
reject.

Figure 1: Polynomial-time algorithm for $(1\text{-}L(N), L(N))$.

Proof. Let us first define two NP sets. The first one is similar to the canonical universal NP complete set except that the number of nondeterministic steps is stated explicitly. For a deterministic Turing machine M, a string x, and $d, t \ge 1$,

$$(M, x, 0^d, 0^t) \in \mathit{UNIV} \iff \begin{array}{l}\text{there exists } w \in \Sigma^d \text{ such that} \\ M \text{ accepts input } (x, w) \\ \text{in at most } t \text{ steps.}\end{array}$$

Obviously, *UNIV* is NP complete. Our second set is defined similarly except that it has, as an additional component, the prefix of an accepting path for the considered machine. For a deterministic Turing machine M, a string x, $d, t \ge 1$, and a string u, where $|u| \le d$,

$$(M, x, 0^d, 0^t, u) \in \mathit{PrefixPATH} \iff \begin{array}{l}\text{there exists } v \in \Sigma^{d-|u|} \text{ such that} \\ M \text{ accepts input } (x, uv) \\ \text{in at most } t \text{ steps.}\end{array}$$

Consider any instance $\tau = (M, x, 0^d, 0^t)$ for *UNIV*. We can define a binary tree T associated with τ as follows. The nodes of T are of the form (τ, u), for $u \in \Sigma^{\le d}$, which are instances for *PrefixPATH*. T's root is (τ, λ) (where λ is the empty string). Clearly, $\tau \in \mathit{UNIV} \iff (\tau, \lambda) \in \mathit{PrefixPATH}$. T's leaves are nodes (τ, u) such that $|u| = d$. A binary string $u \in \Sigma^{\le d}$ is viewed as a path from the root to (τ, u). A string $w \in \Sigma^d$ is called an *accepting path of T* if M accepts input (x, w), or, equivalently, $(\tau, w) \in \mathit{PrefixPATH}$. Clearly, $\tau \in \mathit{UNIV}$ if and only if there exists an accepting path in T.

Let r and e be some integers that will be specified later. Below, we define $r^{\lceil d/e \rceil}$ subtrees T_k of T in such a way that, if there is an accepting path in T, then there exists a subtree T_k that has *exactly one* accepting path. That is,

$$\begin{aligned} \tau \in \mathit{UNIV} &\iff \exists w \in \Sigma^d : \ w \text{ is an accepting path in } T & (3)\\ &\iff \exists k \le r^{\lceil d/e \rceil} : \ T_k \text{ has exactly one accepting path.} & (4) \end{aligned}$$

At this point, we can explain our proof idea, that is, the strategy for deciding whether $\tau \in \mathit{UNIV}$ in deterministic subexponential time. Consider the promise problem $(\mathit{1\text{-}SubTREE}, \mathit{SubTREE})$, where $\mathit{1\text{-}SubTREE}$ is the set of T_k with at most one accepting path, and $\mathit{SubTREE}$ is the set of T_k having an accepting path. $\mathit{SubTREE}$ clearly is an NP set. Then, by Theorem 3.2, this promise problem has a solution in P. Thus, if T_k has exactly one accepting path, we can verify it in polynomial time. Hence, both, equation (3) and (4) give NP-type predicates for deciding whether $\tau \in \mathit{UNIV}$. While there are 2^d possibilities for w in equation (3), we can reduce the scope of k in equation (4) by choosing e large; in other words, while d (binary) nondeterministic guesses are necessary in equation (3), $(d \log r)/e$ guesses are enough when using equation (4). On the other hand, enlarging e will increase the time to decide the promise problem. We will see below that by appropriately choosing e, we can fairly reduce the number of nondeterministic guesses without increasing the time to decide the promise problem too much. That is, the original NP-type predicate is reduced to a simpler one. By iterating this process, we can finally solve the problem without any guesses, i.e. deterministically, and we will see that the whole process can be done in subexponential time.

Let us define the subtrees more precisely. We assign an integer label to each node of T. Subtree T_k of T is then defined as consisting of all nodes having label k and their father nodes. The way to assign labels is therefore crucial. In order to do so, we divide T into *blocks* of depth e. More formally, for each h, where $0 \le h \le \lceil d/e \rceil - 1$, and $u \in \Sigma^{h \cdot e}$, we consider a set $X(\tau, u) = \{ (\tau, uv) \mid v \in \Sigma^e \}$ of nodes in T, which is regarded as a block of depth e.[1] Notice that if $(\tau, u) \in \mathit{PrefixPATH}$, then some elements of $X(\tau, u)$ also belong to $\mathit{PrefixPATH}$. Here, for the decomposition of T satisfying equation (4), we would like to divide $X(\tau, u)$ into $X_1(\tau, u), \ldots, X_r(\tau, u)$ so that if $(\tau, u) \in \mathit{PrefixPATH}$ then some $X_i(\tau, u)$ has exactly one element in $\mathit{PrefixPATH}$. Key point of our proof is that this is possible by using the assumption that $\mathit{PrefixPATH}$ ($\in$ NP) is $\le^{\mathrm{P}}_{\mathrm{btt}}$-reducible to some P-selective set. That is, we have the following lemma.

Key Lemma Let b, $n > 0$ and $r = 6(\lfloor b/2 \rfloor + 1) - 1$. Let L be any set that is $\le^{\mathrm{P}}_{b\text{-tt}}$-reducible to some P-selective set. Then, for any $X \subseteq \Sigma^n$, there exist r disjoint subsets $X_1, \ldots, X_r$ of X with the following property.

$$X \cap L \ne \emptyset \iff \exists i \le r : \ \| X_i \cap L \| = 1.$$

Furthermore, we can compute $X_1, \ldots, X_r$ in polynomial time w.r.t. n and $\| X \|$.

[1] Precisely speaking, when $|u| = (\lceil d/e \rceil - 1)e$ (i.e., $h = \lceil d/e \rceil - 1$), $X(\tau, u)$ should be $\{ (\tau, uv) \mid v \in \Sigma^{d-|u|} \}$. In the following, we omit explaining such exceptional cases.

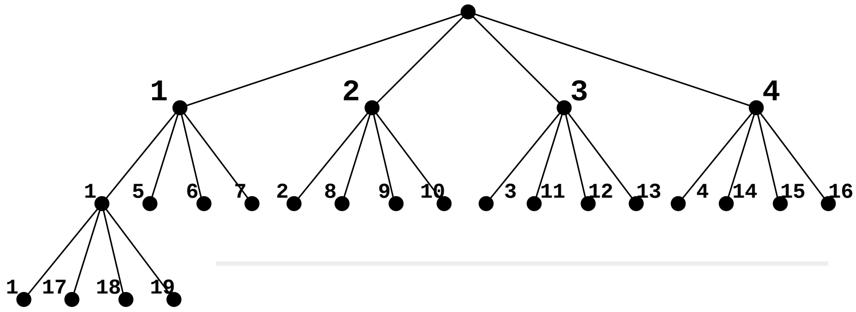

Figure 2: Tree with branching factor 4 and its labeling.

Since *PrefixPATH* is in NP, for some $b > 0$ it is $\leq^{\mathrm{P}}_{b\text{-tt}}$-reducible to some P-selective set by assumption. Thus, from the Key Lemma (with $L = \mathit{PrefixPATH}$ and $X = X(\tau, u)$) we can divide each $X(\tau, u)$ into $r = 6(\lfloor b/2 \rfloor + 1) - 1$ disjoint subsets $X_1(\tau, u), \ldots, X_r(\tau, u)$ of $X(\tau, u)$ such that

$$(\tau, u) \in \mathit{PrefixPATH} \iff$$

$$\exists j \leq r: \ X_j(\tau, u) \text{ has exactly one element in } \mathit{PrefixPATH}.$$

An important point to note here is that r does not depend on e.

The root of T, (τ, λ), gets label 1. Now, let (τ, u) be some node of T, where $u = v_1 v_2 \cdots v_h$, for some $0 \leq h \leq \lceil d/e \rceil$ and $v_1, \ldots, v_h \in \Sigma^e$. All nodes in a set $X_j(\tau, u)$ get the same label. The nodes in $X_1(\tau, u)$ get the same label as (τ, u). For $j > 1$, consider the r-ary tree, where the nodes are the sets $X_i(\tau, w)$. If we go through this tree in a breath first left to right fashion, then the nodes in $X_j(\tau, u)$ get as label the smallest number that not yet occured as a label. Figure 2 provides an example.

More formally, let the *history* of (τ, u) be the sequence $(j_1, \ldots, j_h)$ of indices, where each j_i $(1 \leq i \leq h)$ is the index such that $(\tau, v_1 \cdots v_i) \in X_{j_i}(\tau, v_1 \cdots v_{i-1})$. Note that each history is expressed as a path (from the root to some node) of a r-ary tree.

Let $\mathit{left}(j_1, \ldots, j_h)$ be the number of nodes in the r-ary tree that are in the same depth to the left of the node with history $(j_1, \ldots, j_h)$. That is, $\mathit{left}() = 0$, and

$$\mathit{left}(j_1, \ldots, j_h, j) \ = \ r \cdot \mathit{left}(j_1, \ldots, j_h) + j - 1.$$

Let $v \in \Sigma^e$ and let $(j_1, \ldots, j_h, j)$ be the history of (τ, uv). Then

$$\mathit{label}(\tau, uv) \ = \ \begin{cases} \mathit{label}(\tau, u), & \text{if } j = 1, \text{ and} \\ r^h + (r-1) \cdot \mathit{left}(j_1, \ldots, j_h) + j - 1, & \text{otherwise.} \end{cases}$$

Now, for each k, where $1 \leq k \leq r^{\lceil d/e \rceil}$, define T_k as the subtree of T consisting of all nodes with label k and their father nodes.

It is not hard to show that *label* is computable in polynomial time w.r.t. $|(\tau, u)|$ and 2^e, and furthermore, that the labels are bounded by $r^{\lceil d/e \rceil}$.

Claim 1 *T has an accepting path if and only if for some k, $1 \leq k \leq r^{\lceil d/e \rceil}$, T_k has exactly one accepting path.*

Proof. Since each path of T belongs to one of the subtrees, the if part is obvious.

Assume that T has has an accepting path. Then $(\tau, \lambda) \in \mathit{PrefixPATH}$; hence, by the Key Lemma, some $X_{j_1}(\tau, \lambda)$ has exactly one element (τ, v_1) in $\mathit{PrefixPATH}$. Then since $(\tau, v_1) \in \mathit{PrefixPATH}$, again by the Key Lemma, some $X_{j_2}(\tau, v_1)$ has exactly one element $(\tau, v_1 v_2)$ in $\mathit{PrefixPATH}$. Continuing this argument, we can find $j_1, \ldots, j_{\lceil d/e \rceil}$ and $v_1, \ldots, v_{\lceil d/e \rceil}$ such that each $X_{j_i}(\tau, v_1 \cdots v_{i-1})$ has exactly one element $(\tau, v_1 \cdots v_{i-1} v_i)$ in $\mathit{PrefixPATH}$. In particular, $v_1 v_2 \cdots v_{\lceil d/e \rceil}$ is an accepting path. Thus, T_k, where $k = \mathit{label}(\tau, v_1 v_2 \cdots v_{\lceil d/e \rceil})$, has exactly one accepting path. □ Claim 1

Next, for each $e \geq 1$, consider the following set. For a deterministic Turing machine M, a string x, $d, t \geq 1$, and $1 \leq k \leq r^{\lceil d/e \rceil}$,

$$(M, x, 0^d, 0^t, k) \in \mathit{SubTREE}_e \iff \begin{array}{l} T_k \text{ has an accepting path,} \\ \quad \text{where } T_k \text{ is the subtree of } T \\ \quad \text{defined by } (M, x, 0^d, 0^t) \text{ and } e. \end{array}$$

For each e, clearly $\mathit{SubTREE}_e$ is in NP and thus we could now solve the promise problem $(\mathit{1\text{-}SubTREE}_e, \mathit{SubTREE}_e)$ deterministically in polynomial time applying Theorem 3.2. But we should be careful about the polynomial-time bound, which depends on the choice of e. Precisely speaking, $(\mathit{1\text{-}SubTREE}_e, \mathit{SubTREE}_e)$ has the following upper bound.

Claim 2 *For some polynomial q_{S} and for all $e \geq 1$, there exists a deterministic Turing machine M_e such that*

(i) on inputs of length n, M_e is $q_{\mathrm{S}}(n + 2^e)$-time bounded, and

(ii) $L(M_e)$ is a solution of $(\mathit{1\text{-}SubTREE}_e, \mathit{SubTREE}_e)$.

In other words, for every input $\eta = (M, x, 0^d, 0^t, k)$, M_e halts in $q_{\mathrm{S}}(|\eta| + 2^e)$ steps, and if $\eta \in \mathit{1\text{-}SubTREE}_e$, then $\eta \in \mathit{SubTREE}_e \iff M_e$ accepts η.

Proof. Let $\tau = (M, x, 0^d, 0^t)$ and $\eta = (\tau, k)$. Consider the problem of deciding whether η is in $\mathit{SubTREE}_e$. We can solve this problem by checking whether there exists some $w \in \Sigma^d$ such that 1) M accepts (x, w) in t steps (i.e., w is an accepting path of the tree T defined by τ), and 2) $k = \mathit{label}(\tau, w)$ (i.e., the accepting path w belongs to T_k). Thus, for some polynomial q_1 and for all $e \geq 1$, this can be done nondeterministically in $q_1(|\eta| + 2^e)$ steps. That is,

$\mathit{SubTREE}_e \in \mathrm{NTIME}(q_1(n+2^e))$. Now the claim follows from Theorem 3.2. □ Claim 2

Thus, we reached our goal to reduce the scope of the existential quantifier; that is, $\tau \in \mathit{UNIV} \iff \exists w \in \Sigma^d : w$ is an accepting path in $T \iff \exists k \leq r^{\lceil d/e \rceil} : (\tau, k) \in L(M_e)$. Here, notice that we can easily translate our reduced problem to a new instance for UNIV. Then we can apply the above construction recursively!

Claim 3 *For any e, there exists a deterministic Turing machine $\widehat{M_e}$ such that for every input $\tau = (M, x, 0^d, 0^t)$,*

$$\tau \in \mathit{UNIV} \iff (\widehat{M_e}, \tau, 0^{d'}, 0^{t'}) \in \mathit{UNIV},$$

where $d' = \lceil \log r \rceil \cdot \lceil d/e \rceil$, r as above, and $t' = q_{\mathrm{U}}(|\tau| + 2^e)$, for some fixed polynomial q_{U}.

Proof. Let $\tau = (M, x, 0^d, 0^t)$ and w be string of length at most $d' = \lceil \log r \rceil \cdot \lceil d/e \rceil$. Machine $\widehat{M_e}$ is defined as follows. For a given input (τ, w), $\widehat{M_e}$ simply simulates M_e, the machine defined in Claim 2, on input (τ, w), where w is interpreted as an integer k now. Note that $1 \leq k \leq r^{\lceil d/e \rceil}$. Finally, $\widehat{M_e}$ accepts (τ, w) if and only if M_e accepts (τ, k).

From Claim 1 and Claim 2, we have $\tau \in \mathit{UNIV} \iff \widehat{M_e}$ accepts (τ, w), for some w. Since M_e halts in $q_{\mathrm{S}}(|(\tau, k)| + 2^e)$ steps, $\widehat{M_e}$ halts in $q_{\mathrm{U}}(|(\tau, w)| + 2^e)$ steps, for some polynomial q_{U}. □ Claim 3

Note that although the time bound t increases to t', the crucial point is that the number of nondeterministic steps d' decreases about a factor $(\log r)/e$.

Finally, to show that every NP set L belongs to $\mathrm{DTIME}(2^{n^{O(1/\sqrt{\log n})}})$, let M_L be a deterministic machine and p_L be a polynomial such that for every $x \in \Sigma^*$, $x \in L \iff (M_L, x, 0^{p_L(|x|)}, 0^{p_L(|x|)}) \in \mathit{UNIV}$.

Let x, $|x| = n$, be a string for which we want to decide membership in L. Let $e = \lceil 3\delta(n) \log r \rceil$, where $r = 6(\lfloor b/2 \rfloor + 1) - 1$ and function δ will be chosen appropriately at the end of the proof. (We assume that n is large enough so that $e \geq 3$.) First, define $x_0 = x$, $d_0 = p_L(n)$, $t_0 = p_L(n)$, and $\tau_0 = (M_L, x_0, 0^{d_0}, 0^{t_0})$. For each $i \geq 1$, define inductively $x_i = \tau_{i-1}$, $d_i = \lceil \log r \rceil \cdot \lceil d_{i-1}/e \rceil$, $t_i = q_{\mathrm{U}}(|\tau_{i-1}| + 2^e)$, and $\tau_i = (\widehat{M_e}, x_i, 0^{d_i}, 0^{t_i})$, until $d_i < e$ $(= \lceil 3\delta(n) \log r \rceil)$. Let m be the first integer such that $d_m < e$. Then from Claim 3, we have $\tau_0 \in \mathit{UNIV} \iff \tau_1 \in \mathit{UNIV} \iff \cdots \iff \tau_m \in \mathit{UNIV}$. On the other hand, $x \in L \iff \tau_0 \in \mathit{UNIV}$. Hence, $x \in L \iff \tau_m \in \mathit{UNIV}$. That is, the problem of deciding $x \in L$ is reduced to that of deciding $\tau_m \in \mathit{UNIV}$.

Let us evaluate the deterministic computation time for deciding $\tau_m \in \mathit{UNIV}$. First, we give an upper bound for t_m. Note that for some polynomial p_1, we have $|\tau_i| \leq p_1(t_i)$, for $i = 1, \ldots, m$. Thus,

$$t_m = q_{\mathrm{U}}(|\tau_{m-1}| + 2^e)$$

$$
\begin{aligned}
&\le q_{\mathrm{U}}(p_1(t_{m-1}) + 2^e) \\
&\le q_{\mathrm{U}}(p_1 \circ q_{\mathrm{U}}(\cdots(p_1 \circ q_{\mathrm{U}}(p_L(n) + 2^e))\cdots) + 2^e).
\end{aligned}
$$

Hence, for some constant c_1 and c_2, we have

$$
t_m \;\le\; n^{c_1^m} 2^{c_1^m e} \;=\; 2^{c_1^m(e+\log n)} \;\le\; 2^{c_1^m(c_2\delta(n)\log r+\log n)}.
$$

On the other hand, note that for any $d \ge e \ge 3$, we have $d' = \lceil \log r\rceil \cdot \lceil d/e \rceil \le (3d\log r)/e \le d/\delta(n)$. Thus,

$$
m \;\le\; \log_{\delta(n)} d_0 \;\le\; c_3 \log n/\log\delta(n),
$$

for some constant c_3. Therefore, for some constant c_4,

$$
t_m \;\le\; 2^{c_1^{\frac{c_3\log n}{\log\delta(n)}}(c_2\delta(n)\log r+\log n)} \;\le\; 2^{n^{\frac{c_4}{\log\delta(n)}}(c_2\delta(n)\log r+\log n)},
$$

which takes the smallest order when we choose $\delta(n) = n^{1/\sqrt{\log n}}$. Then, for some constant c_5, we have $t_m \le 2^{n^{c_5/\sqrt{\log n}}}$.

Clearly, "$\tau_m \in \mathit{UNIV}$?" is deterministically decidable in polynomial time w.r.t. $|\tau_m|$. Also τ_m is deterministically computable in polynomial time w.r.t. $|\tau_m|$. Recall that $|\tau_m| \le p_1(t_m)$. Thus, the deterministic computation time for computing τ_m and deciding $\tau_m \in \mathit{UNIV}$ is polynomially bounded by t_m. Therefore, with some constant c_0, it is bounded by $2^{n^{c_0/\sqrt{\log n}}}$. That is, $x \in L$ is deterministically decidable in $2^{n^{c_0/\sqrt{\log n}}}$ steps. □

It remains to prove the Key Lemma.

Key Lemma Let b, $n > 0$ and $r = 6(\lfloor b/2\rfloor + 1) - 1$. Let L be any set that is $\le^{\mathrm{P}}_{b\text{-tt}}$-reducible to some P-selective set. Then, for any set $X \subseteq \Sigma^n$, there exist r disjoint subsets $X_1, \ldots, X_r$ of X with the following property.

$$
X \cap L \ne \emptyset \iff \exists i \le r:\; ||\, X_i \cap L \,|| = 1.
$$

Furthermore, we can compute $X_1, \ldots, X_r$ in polynomial time w.r.t. n and $||\, X \,||$.

Proof. Let g and e be the generator and the evaluator of a $\le^{\mathrm{P}}_{b\text{-tt}}$-reduction from L to a P-selective set A, and let f be a P-selector for A. Define Q to be the set of queries to A for all $x \in X$; that is, $Q = \bigcup_{x\in X} g(x)$. Let $\preceq$ denote $\preceq_{f,Q}$. Recall that $\preceq$ is polynomial-time decidable w.r.t. n and $||\, X \,||$.

For any $u, v \in Q \cup \{\bot, \top\}$, the *interval* $[u, v)$ is the set $\{\, w \in Q \mid u \preceq w \prec v \,\}$. For any set $\mathcal{I}$ of intervals, we simply write $\bigcup \mathcal{I}$ for $\bigcup_{I\in\mathcal{I}} I$.

For each $x \in X$, we can define an associated set of intervals in Q that characterizes the membership of x in L according to a cutpoint of A in Q. More formally, letting $g(x) = \{\, y_1 \preceq \cdots \preceq y_h \,\}$ (where $h \le b$), $y_0 = \bot$, and $y_{h+1} = \top$, we define

$$
\mathcal{I}_x \;=\; \{\, [y_i, y_{i+1}) \mid e(x, g(x), \{y_1, \ldots, y_i\}) = 1, \text{ where } i \in \{0, \ldots, h\} \,\}.
$$

If two adjacent intervals, i.e., $[y_i, y_{i+1})$ and $[y_{i+1}, y_{i+2})$, belong to $\mathcal{I}_x$, we regard them as one interval $[y_i, y_{i+2})$. Note that each $\mathcal{I}_x$ has at most $\lfloor b/2 \rfloor + 1$ intervals.

Let $J_x = \bigcup \mathcal{I}_x$, $J = \bigcup_{x \in X} J_x$, and let z^* be a cutpoint of A in Q. Then, for all $x \in X$, we have $x \in L \iff z^* \in J_x$, and hence, $X \cap L \neq \emptyset \iff z^* \in J$.

By the Combinatorial Lemma stated below, we can select $r = 6(\lfloor b/2 \rfloor + 1) - 1$ subsets $X_1, \ldots, X_r$ of X such that

$$\forall z \in J,\ \exists i \leq r,\ \exists x! \in X_i :\ z \in J_x.$$

Now, we show that $X_1, \ldots, X_r$ have the property claimed in the lemma. Suppose that $X \cap L \neq \emptyset$. Hence, $z^* \in J$. Then, from the above property of $X_1, \ldots, X_r$, there exists some X_i that has exactly one x such that $z^* \in J_x$. This means that X_i has exactly one element (namely, x) in L. (Recall that $x \in L \iff z^* \in J_x$.) Therefore, $|| X_i \cap L || = 1$. □

Combinatorial Lemma Let $\{\mathcal{I}_x\}_{x \in X}$ be any family of sets of intervals in Q, where the index set X is finite, and each $\mathcal{I}_x$ consists of at most ℓ intervals. Let $\mathcal{I}$ be the set of intervals appearing in $\mathcal{I}_x$ for some $x \in X$; i.e., $\mathcal{I} = \{ I \mid I \in \mathcal{I}_x \text{ for some } x \in X \}$. Let $J = \bigcup \mathcal{I}$ and $J_x = \bigcup \mathcal{I}_x$. Then there exist $r = 6\ell - 1$ disjoint subsets $X_1, \ldots, X_r$ of X such that

$$\forall z \in J,\ \exists i \leq r,\ \exists! x \in X_i :\ z \in J_x.$$

Furthermore, if $\preceq$ is polynomial-time computable w.r.t. $\sum_{u \in Q} |u|$, then the selection of $X_1, \ldots, X_r$ can be done in polynomial time w.r.t. ℓ, $|| X ||$, and $\sum_{u \in Q} |u|$.

Proof. First, we construct a minimum size cover of $\mathcal{I}$. We say that $\widehat{\mathcal{I}}$ is a *minimum size cover* of $\mathcal{I}$ if (i) $\widehat{\mathcal{I}} \subseteq \mathcal{I}$, (ii) $\bigcup \widehat{\mathcal{I}} = J$, and (iii) no $\mathcal{I}'$ such that $|| \mathcal{I}' || < || \widehat{\mathcal{I}} ||$ satisfies both, (i) and (ii).

Claim 4 *There is a polynomial-time algorithm that computes a minimum size cover of $\mathcal{I}$.*

Proof. The following greedy algorithm computes a minimum size cover of $\mathcal{I}$.

Minimum-Size-Cover($\mathcal{I}, \preceq$)

$\widehat{\mathcal{I}} \leftarrow \emptyset$; $J' \leftarrow \bigcup \mathcal{I}$;
while $J' \neq \emptyset$ **do**
  $z \leftarrow$ a smallest point in J';
  Select an $I \in \mathcal{I}$ such that $z \in I$ and $|| I \cap J' ||$ is maximal;
  $\widehat{\mathcal{I}} \leftarrow \widehat{\mathcal{I}} \cup \{I\}$; $J' \leftarrow J' - I$;
return $\widehat{\mathcal{I}}$.

Clearly, this algorithm runs in polynomial time. To show its correctness, let $\widetilde{\mathcal{I}} = \{\widetilde{I}_1, \ldots, \widetilde{I}_k\}$ be a minimal cover of $\mathcal{I}$, and let these intervals be in increasing order according to their left endpoints. By $\widetilde{J}_i$ we denote $\bigcup_{j=1}^{i} \widetilde{I}_j$. Let $\widehat{\mathcal{I}} = \{I_1, \ldots, I_h\}$ be the output of Minimum-Size-Cover, where each I_i is selected at the ith iteration of the while-loop. By $\widehat{J}_i$ we denote $\bigcup_{j=1}^{i} I_j$.

Since $\widetilde{\mathcal{I}}$ is a minimal cover, we have $k \leq h$. We will argue that $k = h$, and hence $\widehat{\mathcal{I}}$ is a minimal cover for J as well. Note that both $\widetilde{J}_i$ and $\widehat{J}_i$ are initial segments of J. Therefore, by the choice of I_i, we have $\widetilde{J}_i \subseteq \widehat{J}_i$ for all $i = 1, \ldots, k$, and thus $h \leq k$, since otherwise, $\widetilde{I}$ would not cover J. □ Claim 4

For each $I \in \widehat{\mathcal{I}}$, define *support*$(I)$ to be an x such that $I \in \mathcal{I}_x$, and let *support*$(\widehat{\mathcal{I}}) = \{$ *support*$(I) \mid I \in \widehat{\mathcal{I}}\}$. (If there is more than one x such that $I \in \mathcal{I}_x$, choose one of them for *support*(I).) We will partition *support*$(\widehat{\mathcal{I}})$ into $r = 6l - 1$ groups $X_1, \ldots, X_r$, such that for any two $x, x' \in$ *support*$(\widehat{\mathcal{I}})$:

($*$) if I is an interval in $\widehat{\mathcal{I}}$ with *support*$(I) = x$ and I has nonempty intersection with $J_{x'}$, then x and x' will be in different groups.

Let us first see why property ($*$) of the partitioning $X_1, \ldots, X_r$ satisfies the condition of the lemma. Consider any $z \in J$. Since $\widehat{\mathcal{I}}$ is a cover of J, there is some $I \in \widehat{\mathcal{I}}$ containing z. Let $x =$ *support*(I) and let X_i be the subset containing x. Then, since $J_{x'} \cap I = \emptyset$ for all $x' \neq x$ in X_i, x is the only element of X_i such that $z \in J_x$.

To construct a partitioning of *support*$(\widehat{\mathcal{I}})$ having property ($*$), consider the following undirected (simple) graph $G = (V, E)$.

$$\begin{aligned} V &= \textit{support}(\widehat{\mathcal{I}}), \quad \text{and} \\ E &= \{\{x, x'\} \mid \exists I \in \widehat{\mathcal{I}} : \ \textit{support}(I) = x \text{ and } J_{x'} \cap I \neq \emptyset\}. \end{aligned}$$

Observe that property ($*$) is equivalent to that G is $6l-1$ colorable. To show this property of G, we first consider the following *directed* version $G' = (V, E')$ of G, where

$$E' = \{ (x, x') \mid \exists I \in \widehat{\mathcal{I}} : \ \textit{support}(I) = x \text{ and } J_{x'} \cap I \neq \emptyset \}.$$

Claim 5 *Every vertex of G' has an outdegree of at most $3\ell - 1$.*

Proof. Notice first that every interval in $\mathcal{I}$ intersects with at most three intervals in $\widehat{\mathcal{I}}$, since otherwise, one can define a cover of J that has less elements than $\widehat{\mathcal{I}}$, contradicting the minimality of $\widehat{\mathcal{I}}$. Similarly, every interval in $\widehat{\mathcal{I}}$ intersects with at most two intervals in $\widehat{\mathcal{I}}$. On the other hand, each $x \in V$ has at least one interval in $\widehat{\mathcal{I}}$ and thus at most $\ell - 1$ intervals not in $\widehat{\mathcal{I}}$. Therefore, J_x intersects with at most $3(\ell - 1) + 2 = 3\ell - 1$ intervals in $\widehat{\mathcal{I}}$. □ Claim 5

Claim 6 *Every subgraph of G has a vertex with degree at most $6\ell - 2$*

Proof. Consider any subgraph $\widehat{G} = (\widehat{V}, \widehat{E})$ of G. From Claim 5, it is clear that $\widehat{G}$ has at most $(3\ell - 1) || \widehat{V} ||$ edges; that is, the sum of the degrees of all vertices is at most $2(3\ell - 1) || \widehat{V} ||$. Hence, there is a vertex with degree at most $2(3\ell - 1)$ $= 6\ell - 2$. □ Claim 6

From Claim 6, we derive the crucial property of G.

Claim 7 *G is $6\ell - 1$-colorable. That is, there exists a partition $X_1, \ldots, X_r$ of V, where $r = 6\ell - 1$, such that every X_i forms an independent set in G. Furthermore, some polynomial-time algorithm computes the partition from a given G.*

Proof. We show by induction on the size of V that the simple greedy algorithm that colors vertices in descending order of their degree needs at most $6\ell - 1$ colours. This clearly holds for $|| V || \leq 6\ell - 1$. For larger V, let x be the vertex of G that is colored last by the algorithm and let $\widehat{G}$ be the subgraph of G obtained by deleting x from G. Then, by Claim 6 , we can apply the induction hypothesis to $\widehat{G}$, that is, the algorithm needs at most $6\ell - 1$ colors for $\widehat{G}$. Now, since the degree of x is at most $6\ell - 2$, the algorithm will find a color for x. □ Claim 7

□

Theorem 3.3 can be extended to $\leq^{\mathrm{P}}_{b(n)\text{-tt}}$-reductions, for functions b, as long as b is poly-logarithmically bounded. That is, for $b(n) \leq (\log n)^a$, for some contstant a, if there exists a P-selective set A that is $\leq^{\mathrm{P}}_{b(n)\text{-tt}}$-hard for NP, then NP $\subseteq$ DTIME$(2^{n^{O(1/\sqrt{\log n})}})$.

Acknowledgments

We would like to thank M. Halldórsson of Japan Advanced Institute of Technology and J. Radhakrishnan of Tata Institute of Technology for letting us know that Claim 7 is provable. J. Radhakrishnan also helped us for simplifying the proof of the Combinatorial Lemma. A. Mochizuki of Tokyo Institute of Technology pointed us some error in the earlier version of this paper.

References

[AA94] M. Agrawal and V. Arvind. Quasi-linear truth-table reductions to P-selective sets. *Theoretical Computer Science 158*, 361-370, 1996.

[AHH+93] V. Arvind, Y. Han, L. Hemachandra, J. Köbler, A. Lozano, M. Mundhenk, M. Ogiwara, U. Schöning, R. Silvestri, and T. Thierauf. Reductions to sets of low information content. *Recent Developments in Complexity Theory.* Cambridge University Press, 1993.

(Also available as Technical Report TR-417, University of Rochester, Department of Computer Science, Rochester, NY, April 1992.)

[All86] E. Allender. The complexity of sparse sets in P. In *Proceedings 1st Structure in Complexity Theory Conference,* 1–11, IEEE Computer Society, 1986.

[BDG88] J. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity I.* EATCS Monographs on Theoretical Computer Science, Springer-Verlag (1988).

[BDG91] J. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity II.* EATCS Monographs on Theoretical Computer Science, Springer-Verlag (1991).

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

[BKS95] R. Beigel, M. Kummer, and F. Stephan. Approximable sets. *Information and Computation 120(2)*, 304-314, 1995.

[ESY84] S. Evan, A. Selman, and Y. Yacobi. The complexity of promise problems with applications to public-key cryptography. *Information and Control*, 61:114–133, 1984.

[HHO+93] L. Hemachandra, A. Hoene, M. Ogiwara, A. Selman, T. Thierauf, and J. Wang. Selectivity. In *Proceedings of the 5th International Conference on Computation and Information, ICCI 93*, IEEE, 55-59, 1993.

[HOW92] L. Hemachandra, M. Ogiwara, and O. Watanabe. How hard are sparse sets? In *Proc. 7th Structure in Complexity Theory Conference*, IEEE 222–238, 1992.

[HL94] S. Homer and L. Longpré. On Reductions of NP to Sparse Sets. *Journal of Computer and System Sciences*, 48:324–336, 1994.

[JT95] B. Jenner and J. Torán. Computing functions with parallel queries to NP. *Theoretical Computer Science 141*, 175-193,1995.

[Joc68] C. Jockusch. Semirecursive sets and positive reducibility. *Transactions of the AMS*, 131(2):420–436, 1968.

[KL82] R. Karp, R. Lipton. Turing machines that take advice. *L'Enseignement Mathématique*, 28:191-209, 1982.

[Ko83] K. Ko. On self-reducibility and weak P-selectivity. *Journal of Computer and System Sciences*, 26:209–221, 1983.

[LLS75] R. Ladner, N. Lynch, and A. Selman. A comparison of polynomial time reducibilities. *Theoretical Computer Science*, 1(2):103–124, 1975.

[Mah82] S. Mahaney. Sparse complete sets for NP: solution of a conjecture of Berman and Hartmanis. *Journal of Computer and System Sciences* 25:130-143, 1982.

[O94] M. Ogihara. Polynomial-time membership comparable sets. *SIAM Journal on Computing*, 24(5):1068–1081, 1995.

[OW91] M. Ogiwara and O. Watanabe. On polynomial-time bounded truth-table reducibility of NP sets to sparse sets. *SIAM Journal on Computing*, 20(3):471–483, 1991.

[Sal93] S. Saluja. Relativized limitations of the left set technique and closure classes of sparse sets. In *Proc. 8th Structure in Complexity Theory Conference*, IEEE, 215−222, 1993.

[Sch90] U. Schöning. The power of counting. In *Complexity Theory Retrospective* (A. Selman Ed.), Springer-Verlag (1990), 204−223.

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

[Sel82a] A. Selman. Analogues of semirecursive sets and effective reducibilities to the study of NP complexity. *Information and Control*, 52:36–51, 1982.

[Sel82b] A. Selman. Reductions on NP and P-selective sets. *Theoretical Computer Science*, 19:287–304, 1982.

[Sto77] L. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science* 3:1−22, 1977.

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

[Tor93] J. Torán. Personal Communication.

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

[VV86] L. Valiant and V. Vazirani. NP is as easy as detecting unique solutions. *Theoretical Computer Science* 47:85−93, 1986.

[Wat90] O. Watanabe. Unpublished note.