On the Power of Probabilistic Polynomial Time: $P^{NP[\log]} \subseteq PP$

Lane A. Hemachandra

Gerd Wechsung

September, 1988

CUCS-372-88

On the Power of Probabilistic Polynomial Time: $P^{NP[log]} \subseteq PP$

Lane A. Hemachandra[*]
Department of Computer Science
Columbia University
New York, NY 10027 USA

Gerd Wechsung
Department of Mathematics
Friedrich Schiller University
Jena, East Germany

September, 1988

Abstract

We show that every set in the Θ_2^p level of the polynomial hierarchy—that is, every set polynomial-time truth-table reducible to SAT—is accepted by a probabilistic polynomial-time Turing machine: $P^{NP[log]} \subseteq PP$.

1 Introduction

Comparing the power of various computational paradigms is a core concern of computational complexity theory. In this paper, we study which classes in the polynomial-time hierarchy are contained in probabilistic polynomial time, PP.

1.1 Hierarchical Voting

In a certain Northeastern computer science department, decisions are made in a peculiar way. When an issue is to be decided, a vote is held. Some faculty members are lazy and sleep through the vote. The remaining faculty members vote, some slyly voting many times the same way. After the vote, *the winning position is the position voted for by the most senior faculty member who bothered to vote.* For example, if the chairman votes "yes" on an issue, that is the result of the ballot, even if every other faculty member casts ten "no" votes. We call this scheme *hierarchical voting.*

Some of this work was done while Hemachandra visited Wechsung in Jena.
[*]Research supported by NSF grant CCR-8809174 and a Hewlett-Packard Corporation equipment grant.

1.2 Θ_2^p and PP

In this paper, we show that a Θ_2^p computation (i.e., a $P^{NP[log]}$ computation) can be viewed as a hierarchical vote, and that the hierarchical voting mechanism for Θ_2^p can be implemented by a probabilistic polynomial-time Turing machine.

Theorem $P^{NP[log]} \subseteq PP$.

$P^{NP[log]}$, the class of languages accepted by polynomial-time Turing machines allowed $\mathcal{O}(\log n)$ calls to an NP oracle, was first studied by Papadimitriou and Zachos in 1982 [PZ82]. Recently, after lying dormant for half a decade, the class has taken on new importance. $P^{NP[log]}$, which defines the Θ_2^p level of Wagner's refined polynomial hierarchy[1] [Wag88], has natural complete sets [Kre86,KSW86,Kad87,Wag88], has been shown equivalent to the class of sets truth-table reducible to SAT [Hem87,Wag87], and is the level to which Kadin has collapsed the polynomial hierarchy under the assumption that NP has sparse Turing-complete sets [Kad87].

Our result—$P^{NP[log]} \subseteq PP$—improves a sequence of results of Gill, Papadimitriou and Zachos, Papadimitriou and Yannakakis, and Balcázar, Díaz, and Gabarró. Gill defined PP as the class of languages accepted by probabilistic polynomial-time Turing machines—machines that by definition accept an input if and only if more than half of the probabilistic computation paths accept. Gill exploited the fact that the acceptance probability of PP machines is not bounded away from 1/2 to prove that $NP \subseteq PP$ [Gil77].

Papadimitriou and Zachos showed that the boolean hierarchy—the closure of NP under boolean operations—is contained in $P^{\#P[1]}$ [PZ83]. The results of Gill and Papadimitriou and Zachos were unified and strengthened[2] by Papadimitriou and Yannakakis, and by Balcázar, Díaz, and Gabarró, who showed that D^P ($=_{def} \{L \mid (\exists L_1, L_2 \in NP)[L = L_1 - L_2]\}$) $\subseteq PP$ [PY82], and, indeed, that the boolean hierarchy—the closure of NP under boolean operations [CGH*88]—is contained in PP [BDG88]. However, the technique does not seem useful in proving stronger results.

Since comparing the power of computational paradigms—deterministic, nondeterministic, and probabilistic—is a central concern of computational complexity theory, it seems natural to ask, in the wake of these results, which classes in the polynomial hierarchy are contained in probabilistic polynomial time. Our paper addresses this question.

---

[1] $NP \cup coNP \subseteq \Theta_2^p =_{def} P^{NP[log]} \subseteq \Delta_2^p =_{def} P^{NP} \subseteq NP^{NP} \cap coNP^{NP} \subseteq \cdots$ [Wag88].

[2] Since $PP \subseteq P^{\#P[1]}$; see [Sim75,Sim77] for discussions of the relationships between #P and PP.

2 $P^{NP[log]} \subseteq PP$: Overview of the Simulation

In this section, we prove that $P^{NP[log]} \subseteq PP$.

Theorem 2.1 $P^{NP[log]} \subseteq PP$

Since $P^{NP[log]}$ and the class of sets truth-table reducible [LLS75] to SAT are identical, the latter class is contained in PP.

Lemma 2.2 [Hem87,Wag87] $P^{NP[log]} = \{L \mid L \leq^p_{truth-table} \text{SAT}\}$.

Corollary 2.3 If language L truth-table reduces to an NP set, then L is in PP.

Because of the connection between truth-table reductions, the boolean hierarchy, and $P^{NP[log]}$, our result strengthens the theorem of Balcázar, Díaz, and Gabarró.

Lemma 2.4 ([CGH*88], see also [PZ83]) The boolean hierarchy consists of exactly the sets that are bounded-truth-table reducible to SAT.

Corollary 2.5 [BDG88] The boolean hierarchy—the boolean closure of NP—is contained in PP.

The proof of Theorem 2.1 is by our hierarchical voting model. Given a $P^{NP[log]}$ language, we wish to accept it with a PP machine. The PP machine will have three phases. Phase I is an *elector choosing phase* in which probabilistic computation paths try to guess satisfying assignments to questions that might be asked the oracle during the $P^{NP[log]}$ computation. At the end of the elector choosing phase, each probabilistic path chooses an "elector"—a guess of the $\mathcal{O}(\log n)$ oracle answers. There are $2^{\mathcal{O}(\log n)} = n^{\mathcal{O}(1)}$ electors. Note that some electors may appear many times.

In our faculty analogy, an elector who is not present is a sleeping faculty member, and an elector who appears many times is a faculty member who cheats and casts multiple votes.

It turns out, since finding a satisfying assignment *proves* that a formula is satisfiable but finding an unsatisfying assignment doesn't prove anything, that the elector appearing in Phase I who is lexicographically largest (i.e., represents the largest appearing vector of answers to the queries of $P^{NP[log]}$) will in fact represent the actual answers to the $P^{NP[log]}$ computation.

In Phase II, the *vote fraud phase*, we must *amplify* the power of electors in such a way that each elector c will have the power to defeat all electors c' lexicographically less than c—even if c' received many more votes in Phase I.

Phase II is made up of sub-phases. In the first sub-phase, the votes of the weakest elector (c_0) are not strengthened and the votes of c_1, c_2, $\cdots$ are so strengthened that c_0 is surely the weakest. In the next sub-phase, the votes of c_0 and c_1 are not strengthened, and the votes of c_2, c_3, $\cdots$ are. Repeating this process, eventually the most powerful (lexicographically largest) elector who got any votes has more votes than everyone less powerful added together! During Phase II. a new dummy elector, "silent-majority," is added. This elector may gain a tremendous number of votes—but he will carefully avoid influencing the ballot.

In Phase III, the *vote tallying phase*, the electors state their opinions on the issue: does our $\mathrm{P}^{\mathrm{NP}[\log]}$ language accept the current input? Recall that an elector c_j in fact represents a possibly correct sequence of oracle answers in the computation of $\mathrm{P}^{\mathrm{NP}[\log]}$ on the current input. c_j sees if the P machine of $\mathrm{P}^{\mathrm{NP}[\log]}$ (we will have chosen specific P and NP machines at the start) would accept or reject given the set of answers c_j represents. If it would accept, it flips a coin and votes "accept." If it would reject, it flips a coin and votes "reject." The silent majority paths (which may outnumber even the winning elector) flip coins and vote "accept" on heads and "reject" on tails. Thus the silent majority influences the ballot not a whit, and the vote of the lexicographically largest elector (who is the one that knows the actually answers of the queries of $\mathrm{P}^{\mathrm{NP}[\log]}$ and thus knows if we should accept or reject) carries the ballot.

3 Sketch of Simulation Details

3.1 Phase I: The Elector Choosing Phase

Let $L \in \mathrm{P}^{\mathrm{NP}[\log]}$. Clearly, $L \in \mathrm{P}^{\mathrm{SAT}[\log]}$. Without loss of generality, L is accepted by $M_i^{L(N_j)}$, where N_j is a nondeterministic polynomial-time Turing machine accepting SAT whose computation tree on inputs of size n is always full and of depth exactly n^2, and M_i is a polynomial-time Turing machine that, regardless of the oracle answers it receives: (1) for some polynomial $q(\cdot)$ on inputs of size n asks only queries of length exactly $q(n)$, and that (2) asks (for some k) exactly $k \log n$ queries. (These assumptions are for convenience and follow immediately from the special properties of SAT, and from elementary manipulations.)

We now begin to describe the action of a PP machine, M_{PP}, accepting L. Our PP machine determines what is the actual first query, q_1, to N_j that $M_i^{(\cdot)}(x)$ asks. Then it generates a random path in the computation tree of $N_j(q_1)$. If that path is an accepting path, M_{PP} figures (correctly) that $q_1 \in$ SAT. If the path is *not* an accepting path, M_{PP}

figures (possibly incorrectly) that $q_1 \notin$ SAT. In either case, M_{PP} determines what is the second query that $M_i(x)$ asks its oracle *under the assumption that the answer to the first query is that which we (correctly or incorrectly) believe.* Note that, if the machine on some path determines that $q_1 \notin$ SAT and this is not the case, then the second query it tries will have no relationship to the actual computation—this probabilistic path will be living in a fantasy world.

This process is repeated until we have asked $k \log |x|$ queries. At this point, our probabilistic path has a guess (possibly incorrect) as to the $k \log n$ bit sequence of oracle answers.

Crucially, note that the sequence that is lexicographically largest (viewing "accept" as 1 and "reject" as 0) among those that are chosen by some path is *correct.* Why is this so? Consider the first query. If different probabilistic paths of M_{PP} get different answers, then the "accept" paths are right (as the discovered satisfying assignments assure that q_1 is satisfiable). If $q_1 \in$ SAT, then a path that errs at the start (stumbling on the answer "reject" to the first query), and then does very well on the bogus future queries it wanders through, poses no problem. Its largest possible value is $011\cdots11$, which will be lexicographically less than $1??\cdots??$. Similarly, among those paths that have the first l answers right, a 1 in the $l+1$st position (signifying an accepting path found for the $l+1$st query) outweighs a 0.

The observation that the maximum vector in this structure is the correct one is a technique Krentel [Kre86] uses in his study of optimization problems. However, Krentel used a maximum operator (which was a magical way of computing maximums); our goal is to *implement* the effect of the maximum operator within the framework of probabilistic polynomial computation. This is accomplished by the amplification procedures of Phase II.

3.2 Phases II: The Vote Fraud Phase

Phase II has $n^k - 1$ sub-phases (where k is the constant such that $M_i^{L(N_j)}(\cdot)$ makes $k \log n$ oracle queries). Call the potential electors $c_1 <_{\mathit{lexicographically}} c_2 <_{\mathit{lexicographically}} \cdots <_{\mathit{lexicographically}} c^{n^k}$. In sub-phase one (which has $(q(n))^2$ depth, recall that $q(n)$ is the length of the queries to N_j and that N_j runs in quadratic time), at each of the $(q(n))^2$ steps, if the current elector is silent-majority or $c_i, i > 1$, we flip a coin and keep the same elector. If the current elector is c_1, we flip a coin and keep c_1 as the elector if it yields heads, and make silent-majority our elector if it yields tails.

Consider the extreme case in which, at the start of Phase II, all but one of the probabilistic paths—i.e., $2^{(q(n))^2} - 1$ paths—are votes of c_1, and just one path is a vote of c_2. At the end of sub-phase 1 of Phase II, we still have $(q(n))^2 - 1$ probabilistic paths of M_{PP} as

votes of c_1, but now M_{PP} has $2^{(q(n))^2}$ paths as votes for c_2. So c_2 (and c_3, c_4, $\cdots$, if they occur) have had their voting strength boosted beyond that of c_1.

We continue in a similar fashion. In sub-phase m, $1 < m \leq n^k - 1$, (which has $(q(n))^2$ depth), at each of the $(q(n))^2$ steps, if the current elector is silent-majority or c_i, $i > m$, we flip a coin and keep the same elector. If the current elector is c_i, $i > m$, we flip a coin and keep c_i as the elector if it yields heads, and make silent-majority our elector if it yields tails.

At first, it seems that we have a terrible problem. For example, coming into sub-phase 2, there are $2^{2(q(n))^2}$ probabilistic paths of M_{PP}. *If all but one is a vote of* c_2, *and just one is a vote of* c_3, then since sub-phase 2 is of depth $(q(n))^2$ (and not $2(q(n))^2$), it seems that c_3 does not overpower c_2. The crucial observation is that the italicized phrase above is impossible. During sub-phase 1, c_2 and c_3 are both boosted *proportionally*; thus, coming into sub-phase 2, the worst possible ratio between them is $(q(n))^2 - 1 : 1$, which is a mild enough ratio that our depth $(q(n))^2$ sub-phase 2 deals with it correctly.

3.3 Phase 3: Vote Tallying Phase

This phase is exactly as described in Section 2. The silent majority has no effect on the election, and the strongest elector, who outweights all weaker electors added together thanks to amplification, controls the election. That is, the overall probability of acceptance is greater than 1/2 if and only if the most powerful elector (call her c_{boss}) votes to accept if and only if $M_i^{(\cdot)}(x)$ using the oracle answers represented by c_{boss} accepts if and only if $M_i^{L(N_j)}(x)$ accepts.

4 Conclusions

We have used the model of hierarchical voting to interpret $\mathrm{P}^{\mathrm{NP[log]}}$ computations as ballots that can be simulated by probabilistic polynomial-time machines. It follows that $\mathrm{P}^{\mathrm{NP[log]}} \subseteq \mathrm{PP}$. The techniques used here seem not to extend beyond $\mathrm{P}^{\mathrm{NP[log]}}$; P^{NP} would have an exponential number of electors and thus would need an exponentially long amplification phase. We leave as an open question whether some new technique may prove that P^{NP} is contained in PP.

Acknowledgements

For helpful conversations we are very grateful to Gerhard Buntrock, Zvi Galil, Albrecht Hoene, and Dirk Siefkes.

References

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

[CGH*88] J. Cai, T. Gundermann, J. Hartmanis, L. Hemachandra, V. Sewelson, K. Wagner, and G. Wechsung. The boolean hierarchy I: Structural properties. *SIAM Journal on Computing*, 17(6), 1988. *To appear.*

[Gil77] J. Gill. Computational complexity of probabilistic Turing machines. *SIAM Journal on Computing*, 6(4):675–695, December 1977.

[Hem87] L. Hemachandra. The strong exponential hierarchy collapses. In *19th ACM Symposium on Theory of Computing*, pages 110–122, May 1987.

[Kad87] J. Kadin. $P^{NP[\log n]}$ and sparse Turing-complete sets for NP. In *Proceedings 2nd Structure in Complexity Theory Conference*, pages 33–40, IEEE Computer Society Press, June 1987.

[Kre86] M. Krentel. The complexity of optimization problems. In *18th ACM Symposium on Theory of Computing*, pages 69–76, May 1986.

[KSW86] J. Köbler, U. Schöning, and K. Wagner. *The Difference and Truth-Table Hierarchies for NP.* Technical Report, Fachberichte Informatik, EWH Rheinland-Pfalz, Koblenz, West Germany, July 1986.

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

[PY82] C. Papadimitriou and M. Yannakakis. The complexity of facets (and some facets of complexity). In *14th ACM Symposium on Theory of Computing*, pages 255–260, 1982.

[PZ82] C. Papadimitriou and S. Zachos. *Two Remarks on the Power of Counting.* Technical Report MIT/LCS/TM-228, Laboratory for Computer Science, MIT, Cambridge, MA, August 1982.

[PZ83] C. Papadimitriou and S. Zachos. Two remarks on the power of counting. In *Proceedings 6th GI Conference on Theoretical Computer Science*, pages 269–276, Springer-Verlag Lecture Notes in Computer Science #145, 1983.

[Sim75] J. Simon. On some central problems in computational complexity. January 1975. Ph.D. thesis, Cornell University, Ithaca, N.Y., Available as Cornell Department of Computer Science Technical Report TR75-224.

[Sim77] J. Simon. On the difference between one and many. In *Automata, Languages, and Programming (ICALP 1977)*, pages 480–491, Springer-Verlag *Lecture Notes in Computer Science #52*, 1977.

[Wag87] Klaus Wagner. *Log Query Classes.* Institut für Mathematik 145, Universität Augsburg, Augsburg, W. Germany, May 1987.

[Wag88] K. Wagner. Bounded query computation. In *Proceedings 3rd Structure in Complexity Theory Conference*, pages 260–277, IEEE Computer Society Press, June 1988.