Probabilistic Complexity Classes [1]

by

Alejandro López-Ortiz

Department of Computer Science
Faculty of Mathematics
University of Waterloo

Waterloo, Ontario, Canada, 1991

[1]This report contains the work which was presented to fulfill the requirements of the Master of Mathematics at the Department of Computer Science of the University of Waterloo, plus some additional results included in Chapter 4 (Theorem 4.4)

Abstract

The purpose of this work is to present an overview of the class of problems solvable in probabilistic polynomial time with double sided error (PP). We explore the relationship of PP to other complexity classes, in particular NP and the polynomial hierarchy, and discuss closure under some standard operations such as intersection and complementation. New proofs are given of some results from the literature using techniques developed by the author.

Acknowledgements

Several people made possible the successful completion of this work. Express thanks must be given to some of them.

- To my girlfriend, Claudia Iturriaga Velázquez, for her unconditional love.
- To Prabhakar Ragde. His patience towards my uncommon research style which, in particular, comprises exponentially many queries and interruptions is praised.
- To all my friends, for cheering me up during hard times. Particular thanks to Vladimir Estivill-Castro and Luke O'Connor.

Lastly, financial support from the Institute for Computer Research, the Information and Technology Research Centre, the Department of Computer Science at the University of Waterloo, and the Institute of Mathematics at the National Autonomous University of México is gratefully acknowledged.

Contents

Chapter 1

Introduction

In computational complexity theory, a problem is said to be tractable if there exists an algorithm running in polynomial time on a deterministic Turing machine that solves it [Ed65]. Such problems are known as the class P of problems.

Given a decision problem such that for each positive instance there exists at least one polynomial length string (certificate) from which a correct solution can be derived, it is natural to ask whether there exists any structure to the problem that can be used to find the solution or the certificate string any faster. The problems with a polynomial length certificate form the class NP.

With the introduction of probabilistic Turing machines by Gill [Gil74], several new classes of problems involving probabilistic computations were defined, notably BPP and PP. Briefly, a problem is in BPP if the solution to a given instance can be guessed with more than 75% confidence, and a problem is in PP if the solution can be guessed with more than 50% confidence.

Another class which is closely related to PP is the class $\#P$ of problems with polynomial length certificates together with a count of the number of certificates

for the string. $\#P$ is a counting analogue to NP.

For a given instance, a problem in NP (or $\#P$) can be solved in exponential time by generating all posible strings of polynomial length and verifying (counting) if there is a certificate (how many there are). Sucessful guesses of certificates are called accepting computations.

It can be said that PP is the weakest class of problems that can be approximately *solved* in the intuitive sense of the word. Intuitively, it seems that any problem not in PP will take more than polynomial time to be solved. From this point of view it is interesting to know how well-known classes of problems relate to PP. Along this line we show that, among other relations, $P \subseteq PP$ and $NP \subseteq PP$ (Chapter 3).

The subtle differences between the definitions of P, BPP, and PP lead into classes believed to be far apart, though at this time it is not known if any of the containments $P \subseteq BPP \subseteq PP$ is proper.

A problem L is said to be complete for a class if it belongs to the class and if L is tractable then any other problem in the class is tractable as well. To illustrate some ideas behind each of these classes, we define some properties, give a few problems and locate the smallest of these classes which they are known to belong.

- **2-SAT** is the set of conjunctions of 2-clauses (clauses with two literals) that have a satisfying assignment.
- **Linear Programming** consists of a list of linear inequalities (restrictions) on a set of real-valued variables and a linear function (target). The goal is to find an assignment to the variables that simultaneously satisfy the restrictions and maximize the value of the target function.
- **ZERO** is the set of polynomial expressions which are identically zero.

- **SAT** is the set of boolean formulae which have an assignment rendering them true.
- **MAJ-SAT** is the set of propositional formulas satisfied by a majority of their interpretations.
- **#SAT** is the set of pairs $\langle i, F \rangle$ such that the propositional formula F has more than i satisfying interpretations.
- **Permanent**. Let A be an $n \times n$ matrix. The *permanent* of A is defined as Perm $A = \sum_{\sigma \in \Upsilon} \prod_{i=1}^{n} A_{i,\sigma(i)}$ where Υ is the set of $n!$ permutations of $(1, 2, \ldots, n)$.

From Cook's theorem it follows that 2-SAT is $P-$complete and SAT is $NP-$complete [Coo71]. Khachian showed that Linear Programming is in P. ZERO is in *BPP*, though is not known to be complete for the class. Similarly, Simon [Sim77] showed that MAJ-SAT and #SAT are *PP* complete by noting that several many-one reductions for *NP* problems preserved the number of accepting computations. For the $\#P$ class, Valiant found that computing the permanent of a matrix is a complete problem. This shows that counting the number of accepting computations is a characterization of an interesting class.

These probabilistically solvable classes capture the intuitive idea of many probabilistic approximations to real life problems. For instance, betting schemes should ideally be in *PP*, and solutions given by a MonteCarlo method in *BPP*.

We can define *NP* in terms of $\#P$ as the languages for which the number of accepting computation is at least one. Similarly *PP* is, in terms of $\#P$, the set of languages for which the majority of the computations are either accepting or rejecting, depending upon whether the string is or is not in the language.

Concepts and results are introduced on the oncoming chapters under the scheme:

In the following chapter (2) we define concepts required for the better understanding of this essay.

In chapter 3 we prove $P \subseteq PP$ and $NP \subseteq PP$ using the new ideas of penalty trees and functions.

Then it is shown that $P^{NP[O(\log n)]}$ is in PP (Chapter 4). A new direct proof based upon the techniques of chapter 3 is given. Another simpler proof shows that the set of languages truth-table reducible to NP is contained in PP (Wagner has shown that $P^{NP[O(\log n)]}$ is equivalent to the set of languages truth-table reducible to NP).

In chapter 5 we study closure properties of PP under truth table reductions [BRS90] [For90].

Finally we discuss the result that the PP set of languages is at least as powerful as the polynomial hierarchy (Chapter 6), in other words, $PH \subseteq P^{PP}$ [Tod89].

Chapter 2

Preliminaries

In this essay we use freely the concepts of Turing machine (TM), nondetermininistic Turing machines (NDTM), computation path and computation tree (for nondeterministic and probabilistic machine). A computation path is determined by the sequence of configurations taken by a DTM along a computation, and a computation tree is the equivalent concept for NDTM's. The branching nodes of a computation tree correspond to configurations where the NDTM is in a guess state, and the branches represent each of the possible outcomes of the guess. For a more formal definition of these terms the reader is referred to the standard literature [AHU74], [HU79].

Probabilistic classes have as a model of computation the probabilistic Turing machine, which is a variation on the concept of nondeterminism. More formally,

Definition 1. *A probabilistic Turing machine (PTM) is a Turing Machine with distinguished states called coin-tossing states. For each coin-tossing state, the finite control unit specifies two possible next states. The computation of a probabilistic*

Turing machine is deterministic except that in coin-tossing states the machine tosses an unbiased coin to decide between the two possible next states.

Similarly to nondeterminism, the computation trace of a PTM can be represented as a tree rooted at the starting state and with final states as leaves. Coin-tossing states are those nodes of the computation tree with two children. All other states make for nodes with one child. Any final state is either accepting or rejecting. In nondeterminism, an NDTM has as possible set of computations a tree of computation paths. If at least one of these paths leads to acceptance then it is said that the NDTM accepts the input string. Similarly for PTM's we have the following definitions:

Definition 2. *The probability of a computation path of a PTM is given by* 2^{-m} *where* m *is the number of coin tossing states in the path.*

Thus, the probability of acceptance is naturally defined as the sum of the probabilities over all the accepting paths. We can now introduce the class of problems that can be solved in probabilistic polynomial time :

Definition 3. *The probabilistic polynomial (PP) class is the set of languages* L *for which there is a PTM* M *running in poly-time such that for all words* x*,*

(a) $x \in L \Rightarrow Pr(x \text{ is accepted}) > 1/2$,

(b) $x \notin L \Rightarrow Pr(x \text{ is rejected}) > 1/2$.

Requiring the majority of the answers to be correct is a natural idea. But if the correct solution is given with a probability close to $1/2$ it is hard to differentiate the incorrect solution from the correct one. This gives way to a more restrictive

definition that is also frequently used in real life: bounding away from 1/2 the error margin of the solution.

Definition 4. *The bounded probability polynomial (BPP) class is the set of languages L for which there is a PTM M running in poly-time and some constant $\varepsilon > 0$ such that for all words x,*

(a) $x \in L \Rightarrow Pr(x \text{ is accepted}) > 1/2 + \varepsilon$,

(b) $x \notin L \Rightarrow Pr(x \text{ is rejected}) > 1/2 + \varepsilon$.

Clearly, $BPP \subseteq PP$ since $1/2 + \varepsilon > 1/2$ for $\varepsilon > 0$.

This definition can be modified and still give the same complexity classes. In particular, for *BPP* any constant $\varepsilon \in (0, 1/2)$ gives the same class as the previous definition [Zac79]. For the *PP* class we can also modify the definition to be the set of languages with a poly-time PTM such that $Pr(\text{correct answer}) > 1/2 + \varepsilon(n)$, provided that $\varepsilon(n)^{-1} = \Omega(2^{p(|x|)})$ where $p(\cdot)$ is a polynomial bound on the time taken by the PTM machine.

Note that in the definitions of *PP* and *BPP* we require that the probability of computing the correct answer for each string to be greater than a bound. On the other hand, we could think about amortizing the probability of making a mistake over all the possible inputs or over all the possible inputs up to a certain size. A drawback of this approach is that if the majority of strings in Σ^* are in L, the trivial algorithm which always accepts its input will qualify as a probabilistic way to *solve* this problem. Other possible modifications on the probabilistic bounds of the previous definitions determine more restricted classes of problems.

We may define a class in terms of the exact number of accepting computations. For this we need to augment the machine model in an appropriate manner.

Definition 5. *A counting Turing machine (CTM) is a standard nondeterministic TM with an auxiliary output device that, for a given input, prints in binary notation on a special tape the number of accepting computations induced in the computation tree defined by that input.*

A counting Turing machine M, apart from recognizing a language, computes a function f, with output on the auxiliary device, from the strings over the alphabet Σ into the number of accepting computations of the CTM. Thus a string x is in the language recognized by M if and only if $f(x) \geq 1$.

Definition 6. *$\#P$ is the class of functions that can be computed by counting TMs in polynomial time [Val79]*[1].

BPP and *PP* can also be defined in terms of counting Turing machines and $\#P$.

Definition 7. *A Turing machine is balanced if all the computation paths over a string x are of the same length, and moreover each state is a guess/coin-tossing one.*

Any TM running in polynomial time can be converted into a balanced TM, by adding a clock to count the time taken and, if the machine is nondeterministic or probabilistic, making every state a guess or coin-tossing state. Adding a clock increases by a polynomial amount the time taken by the TM. The outcome of new guess/coin-tossing states can simply be ignored by the TM.

[1] $\#P$ is read as *number P* or *sharp P*. Recently, *number P* seems to be the predominant way.

Definition 8. *The class of problems PP is the set of languages L for which there is a balanced CTM M running in exactly $p(|x|)$ time, for some polynomial p, and computing the function f, such that for all words $x \in \Sigma^*$,*

(a) if $x \in L$ then $f(x) > 2^{p(|x|)-1}$,

(b) if $x \notin L$ then $f(x) < 2^{p(|x|)-1}$.

The definition for *BPP* in terms of $\#P$ is analogous.

Another type of Turing machine that is frequently used is an oracle Turing machine (OTM). An OTM formalizes the idea of free information that could come from a variety of sources, like preprocessed data or expert advice from a human operator.

Definition 9. *An oracle Turing machine (OTM) consists of a standard TM augmented by means of an oracle with an additional oracle tape. An OTM has a special query state, where the oracle, in one step, reads the oracle tape, decides whether the string written in the oracle tape is in its language or not and writes 1 if the string is in the oracle language and 0 otherwise. Thus, the language accepted by the TM is a function of the oracle language.*

The role of key information given along the computation path has been intensively studied. Such information may turn out to be mostly useless for a class of problems but really helpful for a similar class.

An oracle may represent a constant time subroutine call or access to a fixed database, and the oracle answer representing the value of the computation or a successful/unsuccessful lookup.

An oracle Turing machine, making computations of type B, (where B can be any of the classes previously defined, such as P or NP), and asking questions of an oracle with language L is denoted as B^L.

As an example of this strange phenomenon, known as relativization, we have that there exist oracles A and B such that $P^A = NP^A$ but $P^B \neq NP^B$ [BGS75].

Clearly, an OTM running on polynomial time can make at most polynomialy many calls to the oracle. Restricting such OTM to make a lesser number of calls to the oracle seems to alter the power of the OTM and thus leads to different classes of problems. Thus, we denote as $B^{L[k]}$ the set of languages accepted by B computations with at most k questions to an oracle L.

Chapter 3

Relations between P, NP, and PP

The power of the class PP can be analyzed by comparison with other classes. Demonstrating containment of a class by another is interpreted as a relation of powerfulness with the former class being weaker than the latter.

Since any deterministic Turing machine (DTM) is a PTM with no coin-tossing states, we have the following results.

Theorem 1. *$P \subseteq BPP \subseteq PP$.*

The relationship between nondeterministic and probabilistic Turing machines is not so easily observable. At a first sight, the concept of solving a problem unequivocally, with the use of nondeterminism to avoid a possible exponential computation (NP) looks a stronger requirement than obtaining just a probabilistically correct solution in polynomial time. But at a more formal level, the relationship is not so evident. Let us recall that an NDTM accepts a string x if at least *one* computation path accepts x; similarly x is rejected if *no* computation path accepts x. On the other hand a PTM accepts/rejects a string x if the *majority* of the computations

paths accept/reject x. Apparently the latter condition is stronger than the former. The following theorem resolves this situation.

Theorem 2. *$NP \subseteq PP$.*

PROOF. The idea behind this proof is that a known polynomial imbalance in the probabilities of the correct answer can be corrected in polynomial time. That is, an *NP* machine is equivalent to a PTM machine computing correct acceptance with at least $1/2^{p(n)}$ probability, and rejecting with probability one when correct. So the probability of correct acceptance has to be increased to over $1/2$.

Let M be an NDTM machine, and let $L(M)$ be the language accepted by M in time $p(|x|)$ over input x, where $p(\cdot)$ is a polynomial. Without loss of generality, assume that M is a balanced NDTM, over the alphabet $\{0, 1\}$. We construct a modified balanced *PTM* M' from the machine M as follows: replace each of the guessing states by a coin tossing state, with the outcome of a coin toss corresponding to a guess. Each terminal node of the computation tree of M over a string x with $n = |x|$ is reached with probability $1/2^{p(n)}$. A further computation is carried out at each such leaf, which will no longer be terminal in M'. If the terminal leaf of M is rejecting, then the machine will toss a coin $q(n)$ times, where $q(n)$ is a a polynomial no smaller than $p(n)$. If not all of these $q(n)$ tosses are heads, then the machine tosses a coin and accepts/rejects x upon the outcome of the last coin toss. If all the $q(n)$ coin tossings are heads, it throws a coin and rejects x regardless of the outcome. On the other hand, if the terminal leaf of M is accepting, then the machine tosses $q(n) + 1$ coins and accepts regardless of the outcome (this is done to maintain balance, so that computing the probability of acceptance reduces to counting accepting leaves). This concludes the description of M'.

From the definition of M', the probability of rejecting a string that is not in the language L is,

$$\begin{aligned} Pr(x \textit{ is rejected} \mid x \notin L) &= \frac{2^{p(n)}(2^{q(n)}+1)}{2^{p(n)+q(n)+1}} \\ &= \frac{1}{2} + \frac{1}{2^{q(n)+1}} \end{aligned}$$

and the probability of accepting a string in L is at least,

$$\begin{aligned} Pr(x \textit{ is accepted} \mid x \in L) &\geq \frac{(2^{p(n)}-1)(2^{q(n)}-1)}{2^{p(n)+q(n)+1}} + \frac{2^{q(n)+1}}{2^{p(n)+q(n)+1}} \\ &= \frac{1}{2} + \frac{2^{q(n)} - 2^{p(n)} + 1}{2^{p(n)+q(n)+1}} \\ &> 1/2 \qquad\qquad \textit{since } q(n) \geq p(n) \end{aligned}$$

implying that the language L is in PP. □

We call the function $q(n)$ a *penalty function* over the computation path, since it reduces the probability of keeping a result obtained during the computation. We will use penalty functions again in chapter 4.

Chapter 4

Containment of $P^{NP[O(\log n)]}$ in PP

As defined in chapter 2, an oracle Turing machine is a standard Turing machine with the special power of asking questions to an oracle. In this particular chapter we focus on a polynomial time DTM, with $O(\log n)$ queries to an NP oracle.

The capability of accessing an oracle can be used for many different purposes. For instance, a $P^{NP[O(\log n)]}$ machine can use the NP oracle to solve a *co–NP* problem or to make a binary search of an NP property over the vertices of a graph under a particular ordering.

There are several natural problems known to be in $P^{NP[O(\log n)]}$; among these we have UOCOLORING (unique optimum graph coloring), UOVCOVER (unique optimum vertex cover), and UOASAT (unique optimum assignment satisfiability) [Kad89]. Some not so natural problems are known to be complete for the $P^{NP[O(\log n)]}$ class; most such problems involve counting the number of solutions (as shown by Krentel [Kre88]). A complete problem for the $P^{NP[O(\log n)]}$ class similar to SAT is the TREE-SAT problem.

A TREE-SAT is a string representing a balanced labeled tree with nodes labeled

with strings representing SAT instances, edges labeled 0 or 1, and leaves labeled *accept* or *reject*. A particular TREE-SAT instance is accepted if following a path according to the answers to the SAT instances on the nodes of the path given by a SAT oracle leads to an *accept* leaf of the tree.

The importance of this class has been pointed out by work of Kadin [Kad89], Krentel [Kre88] and Papadimitriou [Pap84]. Among other results, it has been shown that under some conditions the polynomial hierarchy can collapse into $P^{NP[O(\log n)]}$ and no further.

In 1988 Hemachandra et al. proved, using concepts and ideas of $\#P$ functions, that $P^{NP[O(\log n)]} \subseteq PP$ [BHW89]. Toda, using a combination of simulation techniques and technical properties of $\#P$ functions, simplified the proof given by Hemachandra [Tod88].

In this chapter we give a new direct proof based on direct simulation of a $P^{NP[O(\log n)]}$ by a PTM machine.

Theorem 3 . $P^{NP[O(\log n)]} \subseteq PP$.

Recall that the i-th question to the oracle is of the form `Is the string` y_i `in the language of the oracle?`. The oracle answer is either 1 meaning yes, or 0 meaning no.

To prove $P^{NP[O(\log n)]} \subseteq PP$, we emulate the precise deterministic computation of the P machine until a query to the oracle is made. At that point a probabilistic approximation to the answer of the oracle is computed by the PTM machine. To avoid compounding the probability of making a mistake, the PTM machine suspends the computation with high probability.

During the simulation of the $P^{NP[O(\log n)]}$ machine by a PP machine the possible answers to a question to the oracle are:

- *sure yes*, in which case the answer of the oracle is 1,
- *perhaps no*, in which case the answer of the oracle is 0, and
- *quit*, in which case the computation is aborted with a 1/2 probability of acceptance and 1/2 of rejecting.

A *sure yes* is obtained when the PTM machine rightly guesses the accepting computation path of a string y_i in the language of the oracle. *Perhaps no* is the result of guessing a rejecting computation path for the string y_i. Since the oracle is an *NP* language, the PTM cannot be certain that y_i is not in the language. *Quit* is a branch taken to represent the uncertainty on a *perhaps no* computation of the PTM machine.

The idea of penalty functions altering the outcome of a computation, as defined in chapter 3, can be further extended to penalty trees.

Definition 10 . *A penalty tree is a balanced tree attached to the end of an specific computation path. In the penalty tree all but two leaves are the result of a coin-tossing deciding whether to accept or reject the input and halting the computation (quit branches); the other two leaves are in the same state as the computation path was before the penalization.*

Without loss of generality we assume that $l(n) = O(\log n)$ questions are always made on input of length n. Thus, we need to embed in the PTM simulation of the computations of M counters for the number of steps and queries to the oracle made by the PTM simulation of the P machine. Whenever those counters exceed the predetermined polynomial time set for the PTM or the $O(\log n)$ queries, the computation needs to be halted with equal probability of accepting/rejecting the input string.

For the purposes of this chapter, we denote the time taken by a $P^{NP[O(\log n)]}$ Turing machine M as $\hat{p}(\cdot)$. In particular $\hat{p}$ bounds the length of the query strings y_i. Similarly the time taken by an NP machine accepting the language of the oracle is denoted as $p(\cdot)$. And preserving the notation of chapter 3, $q(\cdot)$ represents a penalty function as in theorem 2. The input of the machine M is a string x of length n.

To simplify the calculation of probabilities we number the questions to the oracle from $l(n)$ down to 1.

Lemma 1. *The i-th question to the oracle* `Is` $y_i \in L(NP)$`?` *can be replaced by a PTM computation tree of polynomial length* $p(n_i)$, $n_i = |y_i|$, *with the property that if the answer to the question is yes then*

$$\begin{aligned} Pr(\textit{sure yes}) &= \frac{a_i}{2^{p(n_i)}} \\ &\geq \frac{1}{2^{p(n_i)}} \qquad \text{for } 1 \leq a_i \leq 2^{p(n_i)} \end{aligned}$$

$$\begin{aligned} Pr(\textit{perhaps no}) &= \frac{b_i}{2^{p(n_i)} 2^{2^i q(\hat{p}(n))}} \\ &\leq \frac{2^{p(n_i)} - 1}{2^{p(n_i)} 2^{2^i q(\hat{p}(n))}} \\ &< \frac{1}{2^{2^i q(\hat{p}(n))}} \qquad \text{for } 1 \leq b_i \leq 2^{p(n_i)} - 1 \end{aligned}$$

$$Pr(\textit{quit}) = 1 - Pr(\textit{sure yes}) - Pr(\textit{perhaps no})$$

On the other hand if the answer is no then

$$\begin{aligned} Pr(\textit{sure yes}) &= 0 \\ Pr(\textit{perhaps no}) &= \frac{1}{2^{2^i q(\hat{p}(n))}} \\ Pr(\textit{quit}) &= 1 - Pr(\textit{sure yes}) - Pr(\textit{perhaps no}) \end{aligned}$$

PROOF. Let a_i be the number of accepting computations of the NP machine on input y_i and b_i the number of rejecting computations. Modify the proof of theorem 2 with a penalty polynomial $\hat{q}(n) = 2^i q(\hat{p}(n))$, and the result trivially follows. □

Now we have the tools to prove Theorem 3.

PROOF(THEOREM 3). Given the computation path of the $P^{NP[O(\log n)]}$ machine M, we construct a PTM machine M' replacing each of the questions to the oracle with the construction of lemma 1. We want to prove that the machine M' will give the correct answer with more than 1/2 probability.

Since the computation path of M is deterministic, given the answers to oracle questions, a sequence of correctly guessed oracle answers by the PTM machine will lead deterministically into correctly accepting or rejecting the input string.

Hence, the probability of accepting or rejecting correctly the input string of the $P^{NP[O(\log n)]}$ machine is at least the probability of guessing the correct sequence of oracle answers plus the probability of *quit with correct decision* computations. Similarly, the probability of incorrectly finishing the computation is at most the

probability of guessing incorrectly the answers of the oracle plus the probability of *quit with incorrect decision* computations.

We claim that $Pr(\textit{correct answer}) > Pr(\textit{incorrect answer})$, which together with $Pr(\textit{correct answer}) + Pr(\textit{incorrect answer}) = 1$, implies $Pr(\textit{correct answer}) > 1/2$, as required by the definition of *PP*.

Now we compute the *correct answer* worst case probability.

On input x there is a sequence of $l(n)$ questions to the oracle, and even a single incorrect answer from the PTM simulation of the oracle may result on a computation of the $P^{NP[O(\log n)]}$ machine M with the incorrect answer. We make the pessimistic assumption that only the correct sequence of answers from the oracle lead into accepting (if $x \in L(M)$) or rejecting (if $x \notin L(M)$) properly.

This assumption is justified, because if we make a mistake (or many) and still get the correct solution then the probability of obtaining the correct solution increases.

Let $b_{r,s}$ denote the event that questions r through s inclusive are asked an the correct answer is computed (guessed). Let $\overline{b_{r,s}}$ denote the event that questions r through s are asked and an incorrect answer is computed for at least one of them.

Denote by $Pr(b_{r,s})$ the probability of computing the correct answer to oracle queries from question r until s inclusive where $r \geq s$ (recall that questions are numbered in reverse order)

The probability of halting the computation of M' with the correct answer is at least $Pr(b_{l(n),1}) + Pr(\textit{quit with correct decision})$. Similarly, the probability of halting with an incorrect answer is, at most, $Pr(\overline{b_{l(n),1}}) + Pr(\textit{quit with incorrect decision})$. Furthermore, from the construction of the *PP* simulation it follows that,

$$Pr(\textit{quit with correct decision}) = Pr(\textit{quit with incorrect decision})$$

since the *PP* machine tosses a coin before quitting evenly deciding the result of the computation upon the outcome of the coin tossing. But we want to show that the probability of halting with the correct decision is greater than the probability of doing the opposite. Therefore it suffices to show that $Pr(b_{l(n),1}) > Pr(\overline{b_{l(n),1}})$.

Let us estimate the values of the left and right side of the previous inequality. Lemma 1 implies that if a sequence of guessed answers to the oracle is incorrect, the first mistake was necessarily made in a question which had *yes* as correct answer. Hence, either there is a question with correct answer 1 or $Pr(\overline{b_{l(n),1}})$=0.

Let the correct answer to the k-th question be 1. Assume that the first $l(n)-k$ answers from the oracle were computed correctly. When guessing an answer to the k-th question to the the oracle, because of lemma 1, M' guesses a correct answer to it with probability at least $1/2^{p(n_k)}$ and errs with probability at most $1/2^{2^k q(\hat{p}(n))}$.

Thus, we have that the probability of an incorrect result of a complete computation of M' given that the first mistake was made in the k-th question is, at most,

$$\begin{aligned} Pr(\text{incorrect answer} \mid b_{l(n),k+1} \wedge \overline{b_{k,k}}) &\leq Pr(b_{l(n),k+1}) Pr(\overline{b_{k,k}}) \\ &\leq Pr(b_{l(n),k+1}) \left[\frac{1}{2^{2^k q(\hat{p}(n))}}\right] \end{aligned} \tag{4.1}$$

That is, the probability of the first $(l(n)-k)$-th correct answers times the probability of an incorrect 0 on the k-th question.

For the probability of obtaining the correct sequence of $l(n)$ answers from the oracle we have,

$$Pr(b_{l(n),1}) \geq Pr(b_{l(n),k+1}) \left[\frac{1}{2^{p(n_k)}}\right] \prod_{i=1}^{k-1} Pr(b_{k-1,1})$$

$$\geq \quad Pr(b_{l(n),k+1}) \left[\frac{1}{2^{p(n_k)}}\right] \prod_{i=1}^{k-1} \frac{1}{2^{2^i q(\hat{p}(n))}} \tag{4.2}$$

This is the probability of the first $(l(n)-k)$-th answers being computed correctly, multiplied by the probability of a correct 1 on the k-th question, multiplied by the worst case probability of the remaining $k-1$ questions.

So we have,

$$\begin{aligned}
Pr(b_{l(n),1}) &\geq Pr(b_{l(n),k+1}) \left[\frac{1}{2^{p(n_k)}}\right] \prod_{i=1}^{k-1} \frac{1}{2^{2^i q(\hat{p}(n))}} && \text{from (4.1)} \\
&= Pr(b_{l(n),k+1}) \left[\frac{1}{2^{p(n_k)}}\right] \left[\frac{1}{2^{(2^k-2)q(\hat{p}(n))}}\right] \\
&= Pr(b_{l(n),k+1}) \left[\frac{1}{2^{2^k q(\hat{p}(n))}}\right] \left[\frac{2^{2q(\hat{p}(n))}}{2^{p(n_k)}}\right] \\
&\geq Pr(b_{l(n),k+1}) Pr(\overline{b_{k,k}}) \left[2^{q(\hat{p}(n))}\right] && \text{by (4.2) and choice of } q
\end{aligned}$$

We have shown that the probability of M' obtaining the correct answer is much more than the probability of M' making a mistake on the k-th question. It follows that the probability of making a mistake is at most $l(n) \cdot (1/2^{q(\hat{p}(n))}) \cdot Pr(b_{l(n),1})$ which is smaller than the probability of obtaining the correct answer, as required, for the appropriate polynomial $q(n)$. □

Toda extended this result to P machines with a restricted number of positive answers from an NP oracle [Tod88]. Formally, let $Q(M, x, A)$ denote the set of queries made by a DTM machine M with input x to the oracle A.

Lemma 2. *If for all x, $|Q(M, x, A) \cap A| \leq \log n$, then $M^A \in PP$.*

Now we prove that the set of languages which are truth-table reducible to NP is contained in PP and, as a corollary, $P^{NP[O(\log n)]} \subseteq PP$.

Let us define truth-table reducibility. Denote as $\chi_L(x)$ the characteristic function of the set L.

Definition 11. *A language A is truth table reducible (tt-reducible) to a language B if there exists a polynomial time computable function g mapping an input x to the strings $y_1, \ldots, y_k$ and another polynomial time computable function f mapping $\{0,1\}^k$ to $\{0,1\}$ such that $x \in A$ if and only if $f(\chi_B(y_1), \ldots, \chi_B(y_k)) = 1$.*

In simple terms, a language A is tt-reducible to B if, for any instance of A, we can effectively and explicitly determine, before any query to B is made, a polynomial number of queries and a polynomial time formula which tell us how each possible combination of answers to those questions about B determine the answer to the instance of A [Rog67].

If a class C of languages is contained under truth-table reductions in another class E, then E contains the class $P^{C[O(\log n)]}$. (The class of languages accepted by a DTM with $O(\log n)$ questions to a C oracle). This can be easily seen by noticing that there are $2^{O(\log n)}$ possible queries/computations which altogether can be simulated in polynomial time. Thus, the effect of each possible combination of answers can be encoded in a polynomial length boolean formula (provided for us by Cook's Theorem [Coo71]) *before* any question is made.

Theorem 4. *Let A be a language truth-table reducible to NP. Then A is in PP.*

PROOF. Since A is truth table reducible to NP, there exists a P^{NP} Turing machine M_1 accepting A in polynomial time $p(\cdot)$ and asking a predetermined set of questions of the form `Is` y_i `in the language of the oracle?` to the NP oracle. Let M_2 be a balanced NDTM that accepts the language of the oracle in polynomial time $t(\cdot)$ (wlog we assume that t is an increasing function).

On input x with $n = |x|$, we simulate the computation of M_1 with a PTM as follows :

- Compute the sequence of queries to the oracle $\langle y_1, y_2, \ldots, y_k \rangle$.
- For each query to the oracle compute the answer to it using a probabilistic emulation of M_2 (as in lemma (1)) with a penalty tree of depth $q(n) + 1$ where $q(n) > p(n)t(p(n)) + p(n)$. The probability of a *computed accept* given that y_i is in the language of the oracle is at least $1/2^{t(|y_i|)}$. Analogously, the probability of a *computed reject* given that $y \notin L(M_2)$ is equal to $2^{t(|y_i|)+1}/2^{t(|y_i|)+q(n)+1} = 1/2^{q(n)}$.
- In those cases where the answer to the oracle query was decided by a single coin tossing (random halts), the PTM halts the simulation of the computation of M_1, accepting or rejecting the input string x upon the outcome of a final coin tossing.
- In all other cases, the computation continues, keeping as answer to the oracle query the output of the probabilistic simulation.

Notice that apart from the questions to the oracle, the computation of a P^{NP} machine, is deterministic. Hence, a sequence of correctly guessed answers to the oracle queries lead to a correct decision on the simulation of the P^{NP} computation. On the other hand, we pessimistically assume that even a single mistake on the guessed sequence of answers leads to an incorrect outcome of the simulation.

Now we want to prove that the simulation halts with the correct decision with more than 1/2 probability. Equivalently, we want to prove,

$$Pr(\text{correct decision}) > Pr(\text{incorrect decision})$$

Because of the construction of the PTM machine we have that the probability of computing the correct decision is at least the probability of obtaining the correct sequence of answers plus the probability of halting with the correct decision. Similarly the probability of computing the incorrect decision is bounded by the probability of obtaining an incorrect sequence of answers plus the probability of halting with an incorrect decision.

Furthermore, if we recall that penalty trees halt randomly with the same probability for correct and incorrect decisions, we then need only to prove that

$$Pr(\text{correct sequence of answers}) > Pr(\text{incorrect sequence of answers})$$

Let $K_0 = \{i \mid \text{query } i \text{ has answer no}\}$ be the set of queries with correct no answer and $K_1 = \{i \mid \text{query } i \text{ has answer yes}\}$ be the set of queries with correct yes answers. Let p_i denote the probability of computing the correct answer to question i.

The probability of computing the correct sequence of answers (and thus obtaining the correct decision on input x) can be expressed as the product of the probability of obtaining correct yes answers from the oracle, multiplied by the probability of obtaining only correct no answers from the oracle (which are penalized).

$$\begin{aligned} Pr(\text{correct sequence of answers}) = \\ &= \prod_{i \in K_0} p_i \prod_{i \in K_1} p_i \\ &\geq \prod_{i \in K_0} \frac{1}{2^{q(n)}} \prod_{i \in K_1} \frac{1}{2^{t(|y_i|)}} \end{aligned}$$

$$\begin{aligned}
&\geq \left[\frac{1}{2^{q(n)}}\right]^{|K_0|} \prod_{i \in K_1} \frac{1}{2^{t(p(n))}} \\
&= \left[\frac{1}{2^{q(n)}}\right]^{|K_0|} \left[\frac{1}{2^{t(p(n))}}\right]^{|K_1|} \\
&\geq \left[\frac{1}{2^{q(n)}}\right]^{|K_0|} \left[\frac{2^{|K_1|}}{2^{p(n)}}\right] \left[\frac{1}{2^{t(p(n))}}\right]^{p(n)} \\
&\geq \left[\frac{1}{2^{q(n)}}\right]^{|K_0|} \left[\frac{2^{|K_1|}}{2^{p(n)t(p(n))+p(n)}}\right] \\
&> 2^{|K_1|} \left[\frac{1}{2^{q(n)}}\right]^{|K_0|+1}
\end{aligned}$$

As was pointed out in the previous section, the probability of a computed yes when the query string is not in the language of the oracle is 0. Therefore any computed sequence of answers has a computed no answer for all the queries in K_0, and mistakes may only occur where the correct answer from the oracle is yes. An incorrect answer is then a computed no answer which by construction is penalized by a $1/2^{q(n)+1}$ factor. There are at most $2^{|K_1|}$ incorrect sequences of answers, each one with at least one additional computed no answer. We can then bound the probability of computing all the incorrect sequence of answers together:

$$\begin{aligned}
&Pr(\text{incorrect sequence of answers}) \leq \\
&\leq \left[\frac{2^{|K_1|}}{2^{q(n)}}\right] \prod_{i \in K_0} p_i \\
&\leq 2^{|K_1|} \left[\frac{1}{2^{q(n)}}\right]^{|K_0|+1} \\
&< Pr(\text{correct sequence of answers})
\end{aligned}$$

as required. □

Corollary 1 $P^{NP[O(\log n)]} \subseteq PP$.

Chapter 5

Closure Properties of PP

Recently it has been proven that PP is closed under intersection, and union. A complete proof can be found in [BRS90]. We restrict ourselves to an overview of the results.

But first, we show a simpler proof of a closure property.

Lemma 3. *PP is closed under complementation.*

Proof. By definition of PP a PTM M accepting the language $L(M)$, can be modified to a PTM M' identical to M but with the accept state switched to reject and vice versa. Clearly, M' will accept a string in the complement of $L(M)$ with probability greater than $1/2$ and reject a string not in the complement with probability greater than $1/2$. □

The proof of closedness under intersection and union uses techniques related to circuits and gates. First we introduce some terminology.

Definition 12. *A threshold gate is a gate whose output is true if more than half of its inputs are true, and whose output is false otherwise.*

Definition 13. *A PP^{PH}-circuit is a boolean circuit with a threshold as output gate and with ordinary $(\vee, \wedge, \neg)$ circuits as inputs to the threshold gate.*

To simplify computations, throughout this chapter we represent *false* as -1 and *true* as 1 in boolean circuits.

Lemma 4. *Consider an AC_0 circuit of size s and depth $D-1$ with kf output values $x_{i,j}$ with $1 \leq i \leq k$, $1 \leq j \leq f$. Then there exists a PP^{PH}-circuit with inputs $x_{i,j}$ returning true if and only if the polynomial $p(\sum_{1\leq j\leq f} x_{1,j}, \ldots, \sum_{1\leq j\leq f} x_{k,j})$ of degree d, whose coefficients are integers bounded in absolute value by M is positive. Furthermore, the circuit is of size at most $ks + Mf^{O(d)}(d+1)^k$, depth $D+2$, and the fanin for the threshold gate is at most $Mf^{O(d)}(d+1)^k$.*

Proof. Expand the polynomial p and regroup into monomials on the variables $x_{i,j}$ with each monomial having coefficient -1 or 1. The number of monomials in the expansion is at most $Mf^d\binom{d+k}{k}$. Each monomial has degree d or less. The value of each monomial is -1 or 1 and can be computed by a parity circuit of depth two and size $(d+1)2^d$. Therefore there exists a PP^{PH} circuit which determines whether $p(\cdot, \ldots, \cdot)$ is positive or not by looking at the majority of the output from the monomials. □

Since any AC_0 circuit can be simulated by a PTM, we have as a consequence of the previous lemma the following result.

Lemma 5. *Let $N_1, \ldots, N_k$ be NDTM's running on time $t(n)$. Let $acc_N(x)$ (respectively $rej_N(x)$) denote the number of computation paths of N accepting (respectively rejecting) input x. Then for any given rational function $Q(y_1, \ldots, y_k)$ of order d with coefficients bounded by M, there exists a PTM machine M running*

in time $\lceil \log((d+1)^{2k+2}M^2) \rceil + dt(n)$ *such that for all* x, $[acc_M(x) - rej_M(x)] \cdot [Q(acc_{N_1}(x) - rej_{N_1}(x), \ldots, acc_{N_k}(x) - rej_{N_k}(x))]$ *is positive.*

The above lemma implies that the sign of a rational function on the outcome of k polynomial time NDTM's as variables can be computed in time exponential on the number of machines plus a polynomial factor. Hence if the number of NDTM's is a constant for all inputs x, the sign of the rational function can be computed in polynomial time.

Now consider the rational functions,

$$\begin{aligned} P_n(x) &= (x-1)\prod_{i=1}^{n}(x-2^i)^2 \\ Q_n(x) &= (-1/2)(P_n(x)+P_n(-x)) \\ A_n(x,y) &= 2P_n(x)Q_n(y) + 2P_n(y)Q_n(x) + Q_n(x)Q_n(y). \end{aligned}$$

It can be easily shown that for any integral values $(|x|, |y|) \in [1, 2^n] \times [1, 2^n]$, the function $A_n(x, y)$ is positive if and only if x and y are positive. Moreover, since $A_n(x, y)$ is a rational function of order $O(n)$ with coefficients of order $O(2^{n^2})$, it follows from lemma (4) that the sign of $A_n(x, y)$ can be computed in probabilistic polynomial time.

Thus, given two NDTM's N_1, N_2 there exists a PTM M with a majority of accepting paths if and only if the rational function $A_n(x, y)$ is positive, which in turn is positive if and only if $(acc_{N_1}(x) - rej_{N_1}(x)) \cdot (acc_{N_2}(x) - rej_{N_2}(x))$ is positive.

But this implies that M accepts x if and only if x is in the languages accepted by the PTM's N_1', N_2' [1].

[1] Recall that any NDTM is in particular a PTM.

A simple extension on the definition of the function $A_n(x, y)$ provides us with similar proofs of closedness of PP under polynomial time bounded depth reductions, poly-time conjunctions or disjunctions, threshold reductions and some other reductions (see [BRS90]).

Recently, Fortnow and Reingold showed the following theorem [For90].

Theorem 5. *PP is closed under truth-table reductions.*

The proof given by Fortnow and Reingold uses similar techniques to those used by Beigel et al. [BRS90] and is thus omitted.

Corollary 2. $P^{PP[O(\log n)]} \subseteq PP$.

Up to this point the proof of containments in PP relied heavily and mostly on knowing in advance the set of questions and the effect a sequence of answers on the computation. This result by Fortnow and Reingold shows that, even for a PP language as oracle, having beforehand knowledge of the sequence of questions and the consequences of its answers from the oracle on the deterministic computation suffices for a PP simulation to succeed.

Chapter 6

$PH \subseteq P^{PP}$

The polynomial hierarchy PH was first defined by Stockmeyer [Sto76]. Consider the sequence of classes $\Sigma_k^P,\ k = 1, 2, \ldots$ where $\Sigma_0^P = P$ and $\Sigma_{i+1}^P = NP^{\Sigma_i^P}$. The PH class of languages is defined as $PH = \cup_{k \geq 0} \Sigma_k^P$. Some properties of PH are, $\Sigma_1^P = NP$ and $\Sigma_k^P \subseteq \Sigma_{k+1}^P$ for all k. Also, if $P = NP$ then $P = PH$.

In this chapter we sketch Toda's result that $PH \subseteq P^{PP}$. The idea of this proof is to build a chain of small containments starting on PH and finishing in P^{PP}. First we introduce the operators $\oplus$, BP, and P to create new classes out of old ones. Then we show that $PH \subseteq \mathsf{BP} \oplus P$. It turns out to be a trivial fact that $\mathsf{BP} \oplus P \subseteq \mathsf{P} \oplus P$. Then we prove the containment $\mathsf{P} \oplus P \subseteq P^{PP}$, which implies $PH \subseteq P^{PP}$.

For the scope of this section, a Q-certificate for a string x with respect to the language L is a string w of polynomial size on the length of x, such that x is in L if and only if (x, w) belongs to a language of type Q.

The parity polynomial $\oplus P$ class is the set of languages L for which there is an NDTM M running in polynomial time such that for all words x, x is in L if and only if the number of accepting computations (or certificates) of M on input x is

odd.

The concepts of bounded probability, probability and parity can be extended to any class of languages. Namely, given a class C of certificate languages, and a language L, we define $\oplus C$, $\mathsf{BP}C$, and $\mathsf{P}C$ as follows.

- $L \in \oplus C$ if and only there exists a language of certificates $A \in C$ and a polynomial p such that $x \in L$ if and only if the number of certificates of length $p(|x|)$ in A is odd.
- $L \in \mathsf{BP}C$ if and only there exists a language of certificates $A \in C$ and a polynomial p such that if $x \in L$ then the number of certificates of length $p(|x|)$ in A is at least $\frac{2}{3}2^{p(|x|)}$. If $x \notin L$ then the number of certificates of x is at most $\frac{1}{3}2^{p(|x|)}$.
- $L \in \mathsf{P}C$ if and only there exists a language of certificates $A \in C$ and a polynomial p such that $x \in L$ if and only if the number of certificates of length $p(|x|)$ in A is greater than $\frac{1}{2}2^{p(|x|)}$.

Lemma 6. *PH* $\subseteq \mathsf{BP} \oplus P$

Proof(sketch). First we show that a one level difference on the polynomial hierarchy can be closed using the composition of operators $\mathsf{BP}\oplus$. In other words, $\Sigma_k^p \cup \Pi_k^p \subseteq \mathsf{BP} \oplus \Pi_{k-1}^p$. The proof of $\Sigma_k^p \subseteq \mathsf{BP} \oplus \Pi_{k-1}^p$ goes as follows. From the definition of the polynomial hierarchy it follows that any language $L \in \Sigma_k^p$, has a language of Π_{k-1}^p-certificates for it. Thus if we can guess an odd number of certificates with bounded probability that will show the containment of Σ_k^p in $\mathsf{BP} \oplus \Pi_{k-1}^p$.

Instead of attempting to guess an odd number of certificates, which is hard since certificates are guessed independently, the simulation randomly attempts to construct a related problem that has an odd number of certificates (a unique certificate, to be precise) if and only if the original problem has at least one certificate.

It has been shown by Valiant and Vazirani [Val85] that there is a probabilistic technique to construct such a problem with probability of success greater than 1/4. Moreover, for this particular case, such problem can only be constructed if the number of certificates is odd and a success in the construction can be verified on polynomial time.

Hence, a simple iteration on the random construction of the problem increases the probability of success over 2/3. So, for any problem in Σ_k^p, an family of problems in Π_{k-1}^p with an odd number of certificates can be probabilistically obtained with probability higher than 2/3.

Finally, since any BP class of languages is closed under complementation, we have the result, $\Sigma_k^p \cup \Pi_k^p \subseteq \mathsf{BP} \oplus \Pi_{k-1}^p$. In other words, the k-th level of the polynomial hierarchy is contained in the $k-1$-th parity-randomized level.

It can be shown that $\mathsf{BP} \oplus \mathsf{BP} \oplus P \subseteq \mathsf{BP} \oplus P$. Hence, using induction over the levels of the polynomial hierarchy we have,

Basis of induction : Trivially $\Sigma_0^p \subseteq \mathsf{BP} \oplus P$.

Induction step : Assume $\Sigma_{k-1}^p \subseteq \mathsf{BP} \oplus P$, then $\Sigma_k^p \subseteq \mathsf{BP} \oplus \mathsf{BP} \oplus P$, which has been pointed out to be $\mathsf{BP} \oplus P$.

This implies $PH \subseteq \mathsf{BP} \oplus P$. □

From the definition of BP and P and lemma 6 we have, as a corollary, $PH \subseteq \mathsf{P} \oplus P$.

Lemma 7. $\mathsf{P} \oplus P \subseteq P^{PP}$

PROOF(SKETCH). It turns out that a *PP* machine can simulate in parallel a P operator and $\oplus P$ machine. In other words, given a string x in a language $L \in \mathsf{P} \oplus P$, the total number of accepting computations of this particular *PP* machine is of the form $a \cdot 2^q + b$, where b is the number of certificates of x in $\oplus P$. The factor $a \cdot 2^q$ is the result of spurious accepting computations arising from the verification of certificate candidates in the $\oplus P$ language of certificates.

Consider a second *PP* machine which on input (x, k) emulates the computation of the above machine but penalizes accept states according to the factor k in such a way that the majority of the computations are accepting after the penalization if and only if more than $k\%$ computations were accepting originally.

Thus the actual number of certificates of the string x can be obtained using binary search with $O(\log n)$ calls to such *PP* oracle. The string x is accepted by the P^{PP} machine if such number of accepting states modulus 2^q is greater than $(1/2)2^{p(|x|)}$, where $p(|x|)$ is the polynomial length of the certificate strings. Therefore $\mathsf{P} \oplus P \subseteq P^{PP}$. □

Theorem 6. *PH* $\subseteq P^{PP}$.

PROOF. Follows from lemma 5 and 6. □

The implications of this result are varied. In particular we have that, unless the polynomial hierarchy collapses to finitely many levels, any language in the polynomial hierarchy is Turing reducible to a *PP*-complete language (e.g. **MAJ-SAT**). In other words, if *PP* $\subseteq$ *PH* then the polynomial hierarchy collapses to finitely many levels.

Chapter 7

Conclusion

The classes NP, BPP and PP are so similarly defined, that it is natural to expect a problems of about the same complexity to be contained in all of them. Nevertheless, as the results in this essay point out, these classes are very different (unless of course $P = NP$).

The simulation techniques and penalty functions proposed in this essay provide more insight on the ways a PTM can emulate a different type of TM. These simulation techniques may suggest new relationships among complexity classes and provide insight for further research.

We conjecture that the inverse of the function $A_n(x, y)$ as described in chapter 5, can be used as a penalty function to prove by direct simulation the closedness of PP under intersection. If that is the case, the result most likely can be extended to closure under polynomial time truth-table reductions.

Whether $PH \subseteq PP$ remains to be proven, and it is widely conjectured to be false [Joh89]. The author believes, based on recent advances, that such containment is not so unlikely to be true.

In particular, if a Σ_k^P complete or a Σ_k^P hard problem is contained in PP then the polynomial hierarchy up to the k-th level is contained in PP. So far this containment is known only for the first level of the polynomial hierarchy.

Bibliography

[AHU74] A. V. Aho, J. E. Hopcroft and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.

[BGS75] T. Baker, J. Gill, and R. Solovay, Relativizations of the $P =?NP$ question, *SIAM Journal of Computing*, Vol. 4, 1975, 1143-1154.

[BHW89] R. Beigel, L. Hemachandra, and G. Wechsung, On the power of probabilistic polynomial time : $P^{NP[log]} \subseteq PP$, *Proceedings of the 4th Structure in Complexity Theory Conference*, IEEE, 1989, 225-227.

[BRS90] R. Beigel, N. Reingold, and D. Spielman, *PP is Closed under Intersection*, Yale University Technical Report, Department of Computer Science, YALEU/DCS/TR-803, July, 1990.

[Coo71] S. A. Cook, The Complexity of theorem-proving procedures, *Proceedings of the 3rd ACM Symposium on Theory of Computing*, 1971, 151-158.

[Edm65] J. Edmonds, Paths, trees, and flowers, *Canadian Journal of Mathematics*, Vol. 17, 1965, 449-467.

[For90] L. Fortnow and N. Reingold, *PP is Closed Under Truth-Table Reductions*, University of Chicago Technical Report, Department of Computer Science, 90-30, 1990.

[GJ79] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP−completeness*, Freeman, New York, 1979.

[Gas86] W. Gasarch, *The Complexity of Optimization Functions*, Technical Report 1652, Dept. of Computer Science, University of Maryland, 1986.

[Gil74] J. Gill, Computational complexity of probabilistic Turing machines, *Proceedings of the 6th ACM Symposium on Theory of Computing*, 1974, 91-95.

[Gre90] F. Green, An oracle separating $\oplus P$ from PP^{PH}, *Proceedings of the 5th Structure in Complexity Theory Conference*, IEEE, 1990, 295-298.

[Har90] J. Hartmanis, New developments in structural complexity theory, *Theoretical Computer Science*, Vol. 71, 1990, 79-93.

[HU79] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, Princeton, NJ, 1979

[Joh89] D. S. Johnson, *A Catalog of Complexity Classes*, manuscript, 1989.

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

[Kre88] M. W. Krentel, The complexity of optimization problems, *Journal of Computer and System Sciences*, Vol. 36, 1988, 490-509.

[Lóp90] A. López-Ortiz, *Probabilistic Complexity Classes*, M.Math. Thesis, Department of Computer Science, University of Waterloo, 1990.

[Pap84] C. Papadimitriou, On the complexity of unique solutions, *Journal of the Association of Computing Machinery*, Vol. 31, 1984, 392-400.

[Rog67] H. Rogers Jr., *Theory of Recursive Functions and Effective Computability*, McGraw-Hill, New York, 1967.

[Sim77] J. Simon, On the difference between one and many, *Proceedings of the 4th International Colloquium on Automata, Languages and Programming*, Springer-Verlag, Berlin, 1977, 480-491.

[Sto76] L. Stockmeyer, The polynomial time hierarchy. *Theoretical Computer Science*, Vol. 3, 1976, 1-22.

[Sto87] L. Stockmeyer, Classifying the computational complexity of problems, *Journal of Symbolic Logic*, Vol. 52, 1987, 1-43.

[Tod88] S. Toda, On probabilistic computations with restricted access to *NP* oracles, *manuscript*, 1988.

[Tod89] S. Toda, *PP* is $\leq_T^p$-hard for the polynomial time hierarchy, *Proceedings of the 30th Symposium on Foundations of Computer Science*, 1989, 514-519.

[Val79] L. G. Valiant, The complexity of computing the permanent, *Theoretical Computer Science*, Vol. 8, 1979, 189-201.

[Val85] L. G. Valiant and V. V. Vazirani, *NP* is as easy as detecting unique solutions, *Theoretical Computer Science*, Vol. 47, 1986, 85-93.

[Zac79] S. Zachos, Robustness of probabilistic computational complexity classes under definitional perturbations, *Information and Control*, Vol. 54, 1982, 143-154.