MIT/LCS/TM-228

TWO REMARKS ON THE POWER OF COUNTING

Christos H. Papadimitriou

Stathis K. Zachos

August 1982

TWO REMARKS ON THE POWER OF COUNTING

Christos H. Papadimitriou and Stathis K. Zachos
Laboratory of Computer Science, M.I.T., Cambridge, USA

August 1982

Abstract: The relationship between the polynomial hierarchy and Valiant's class #P is at present unknown. We show that some low portions of the polynomial hierarchy, namely deterministic polynomial algorithms using an NP oracle at most a logarithmic number of times, can be simulated by one #P computation. We also show that the class of problems solvable by polynomial-time nondeterministic Turing machines which accept whenever there is an *odd* number of accepting computations is idempotent, that is, closed under usage of oracles from the same class.

Keywords: Counting problems, oracle computation, polynomial hierarchy, parity problems, machine simulation.

1. Introduction.

Counting Turing machines, and the class #P of counting problems that can be solved by such machines in polynomial time, were first introduced and studied by Valiant [Val]. The relationship of #P with other complexity classes has since been an intriguing open question. It is well-known that NP is contained in #P (we use the term #P, originally meant for a class of functions, to denote the class of *languages* which are accepted in polynomial deterministic time with one invocation of a #P computation). Furthermore, Simon [Si] showed some close relationships between #P and the probabilistic class PP [Gi]. On the other hand, it was suspected that #P lies above the whole polynomial hierarchy [St], but no proof of this is known to date. Angluin [An] showed that #P, appropriately relativized, is more powerful than $\Sigma_2 P$.

We were interested in showing that, in the unrelativized case, the class $\Delta_2 P = P^{NP}$ is contained in #P. What we were able to show is that a portion of $\Delta_2 P$, namely the class $P^{NP[\log]}$ of problems solvable by polynomial-time algorithms using a logarithmic number of calls to an NP oracle, is indeed contained in #P. This class contains, for example, the problem of testing whether a given undirected graph has a unique optimum clique [Pa] (it is not known, however, whether this problem is complete for $P^{NP[\log]}$). It also contains the class D^P [PY] of languages that are the intersection of a language in NP and one in coNP. Furthermore, there is a whole hierarchy of classes of languages definable as NP predicates combined by Boolean connectives. The *i*th level of this hierarchy consists of all languages that can be expressed as the union of *i* languages in D^P. The limit of this hierarchy turns out to be identical to the class of languages recognized by polynomial algorithms using a *bounded* number of calls to an NP oracle, and thus is also a subset of #P, by our result.

The technique used in the proof of this theorem employs a simple way of encoding a computation with oracle branchings into the number of accepting computations of a single nondeterministic computation, which can then be computed by a counting Turing machine, and then decoded. This technique seems to be useful in simulations by counting machines. Using a variant of this technique, we show an interesting property of the class $\oplus P$ of problems that can be solved by Turing machines which accept if the number of accepting computations is odd. A typical (complete) problem in $\oplus P$ is the set of all graphs that have an odd number of Hamilton circuits. $\oplus P$ can be considered as a more moderate version of the counting idea. The relationship between $\oplus P$ and NP is not known, although it is suspected that NP is weaker [Va2]. What we show is that $\oplus P^{\oplus P} = \oplus P$, and thus $\oplus P$ appears to behave differently from NP. This fact had been proved independently by Valiant [Va2].

2. Definitions

For basic Turing machine definitions see [GJ, LP]. All computation paths of a nondeterministic Turing machine on a particular input (or the computations from *any* given configuration) form a tree. We do not insist that all leaves of this tree have the same depth (this can be achieved by a variety of padding techniques). We shall need to define certain notation for manipulating nondeterministic computation trees. If $C_1, \ldots, C_k$ are configurations of a Turing machine (equivalently, the computation trees starting from these configurations), then CHOOSE$(C_1, \ldots, C_k)$ denotes the computation tree consisting of a new root, which has nondeterministic branches to all these computations. Also, APPEND(C_1, C_2) denotes the computation tree consisting of the computation tree of C_1 with a copy of C_2 hanging from each accepting leaf of C_1. Finally, DUPL$_n$ denotes a computation tree which has exactly n accepting leaves.

If M is a nondeterministic Turing machine, we let COUNT(M,x) denote the number of accepting computations of M on input x. Thus, NP can be defined as the class of languages L for which there is a nondeterministic Turing machine M such that $L = \{x \mid \text{COUNT}(M,x) > 0\}$. #P is the class of languages that can be recognized by a deterministic polynomial-time algorithm which uses *only once* an oracle computing COUNT. $P^{NP[\log]}$, a subclass of $\Delta_2 P = P^{NP}$, is the set of problems solvable by deterministic polynomial-time algorithms which use an oracle in NP a number of times which is at most proportional to the logarithm of the length of the input of the algorithm. Finally, $\oplus P$ is the class of all languages L, for which there exists a nondeterministic Turing machine M such that $L = \{x \mid \text{COUNT}(M,x) \textbf{ mod } 2 = 1\}$.

3. The Main Theorem

In this Section we prove the following:

Theorem 1 $P^{NP[\log]} \subseteq \#P$.

Proof Suppose that L is a language recognized by a deterministic Turing machine M with an oracle in NP, so that at most $p(|x|)$ steps, and at most $\log(|x|)$ oracle steps are used in the computation on input x. Assume without loss of generality that the oracle queries are of the form (M',x'), asking whether a nondeterministic Turing machine M' accepts an input x' in time $p(|x|)$. We also assume that M always asks *exactly* $\lfloor \log |x| \rfloor$ queries. We shall design a deterministic polynomial algorithm which decides L by using one computation of COUNT.

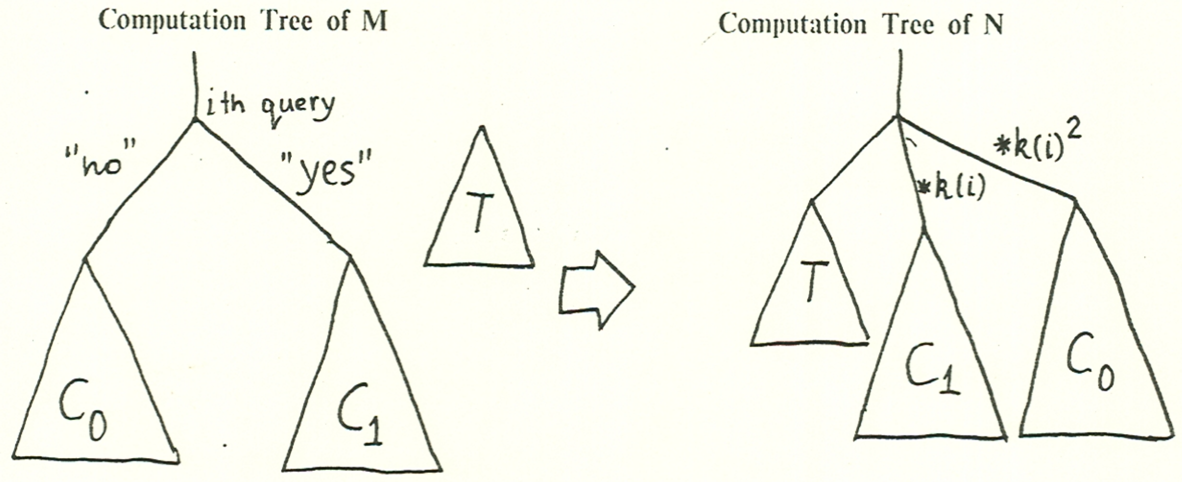
The idea is the following: By multiplying the number of leaves in different subtrees of the computation tree of M by exorbitantly large numbers, we can unambiguously encode the outcomes of all the oracle calls along all possible computation paths of M.

(Notice that M, although deterministic, has a polynomial number of computation paths, due to the oracle steps.)

The algorithm first constructs, based on M and x, a nondeterministic Turing machine N, as follows. N is programmed basically like M, except in the query configurations. If the query (M',x') is the ith query asked by M, and M goes to configuration C_1 if the answer is "yes", and C_0 if "no", then N executes the nondeterministic program shown below:

CHOOSE(ID_0(M',x'),APPEND($DUPL_{k(i)}$,C_1),APPEND($DUPL_{k(i)^2}$,C_0)),

where ID_0(M',x') denotes the initial configuration of the machine M on input x), and the k(i)'s are integers to be defined later. In words, N nondeterministically chooses among three possibilities: Either to simulate M' on x' and stop; or to assume the answer is "yes" and amplify the subsequent computation by a factor of k(i); or finally to assume the answer is "no", in which case the answer is amplified by $k(i)^2$ (see the figure below for a pictorial presentation of the construction using computation trees).



*i*th Query: Does T have an Accepting Leaf?

The full algorithm is the following:

```
ALGORITHM A;
begin
construct N as described above;
n ← COUNT(N,x);
i ← 1;
while n > 1 do
 begin
 if n mod k(i) = 0 then n ← n div k(i)^2
                  else n ← (n mod k(i)^2) div k(i);
 i ← i + 1
 end;
if n = 1 then accept else reject
end .
```

The k(i)'s are defined by the following equations:

$$k(\lfloor \log |x| \rfloor) = 2^{p(|x|)}$$

$$k(i) = k(i+1)^4$$

Thus, k(i) increases rapidly as i decreases (or, equivalently, as we proceed from later to earlier queries). The maximum value of k(i) is k(1), which is $2^{p(|x|)|x|^{2}/4}$. Notice that this number is of length polynomial in |x|, and therefore our construction produces a nondeterministic Turing machine N which still obeys a polynomial bound on the depth of all computations. It is this growth of the k(i)'s, a necessary ingredient for our arguments, which limits the applicability of our proof to the case of logarithmically many queries.

We shall argue that the algorithm above accepts x iff M does. We shall show, by induction on i, that the number n in the beginning of the ith iteration of the main loop of the algorithm A denotes the number of accepting leaves of the computation subtree of N, which corresponds to the computation subtree of M starting at the *i*th query. This certainly holds for $i = 1$. Assume now that it holds after the first $i - 1$ iterations. Suppose that the *i*th query has answer "no". This means that there is no accepting path of M' on the queried x'. Since such paths are the only paths in the subtree of N which are not multiplied by a multiple of k(i), and since the number of such paths cannot exceed $k(i) - 1$, this is equivalent to saying that the value of n before the ith iteration (by induction hypothesis the total number of such paths) is divisible exactly by k(i). Thus the **then** branch of the test is taken, and the new value of n is n **div** $k(i)^2$, which is exactly the number of accepting computations in the subtree of N corresponding to C_0. This is because the growth of the k(i)'s is such that the total number of leaves in the subtree of N corresponding to C_0 (or for that matter to C_1) is less than k(i). Similarly, if the answer to the *i*th query is "yes", then the **else** branch is taken, and n becomes the number of accepting leaves in a subtree of N that corresponds to C_1. The induction is complete.

Therefore, after the last execution of the loop, the value of n is the number of accepting computations of M after the last oracle step. This number is of course 1 or 0, depending on whether x is accepted by M or not. Q.E.D.

Notice that we can prove the same way a stronger result, namely that #P can simulate a logarithmic number of queries from PP. PP is defined as the class of languages L for which there is a nondeterministic Turing machine M such that all computations of M on input x have length exactly p(|x|) for some polynomial p(.), and $L = \{x \mid \text{COUNT}(M,x) > 2^{p(|x|)-1}\}$. PP is known to contain NP [Gi].

4. Parity Counting

A variant of the proof technique of the previous Section can be used to show the following result:

Theorem 2 $\oplus P^{\oplus P} = \oplus P$.

Proof: Let L be a language recognized by a parity machine M which uses as an oracle another parity machine M). We shall design a parity machine N which accepts L with no oracles. N is programmed exactly like M, except in the oracle steps. Suppose that M is at an oracle step with query (M',x'), and let C_1 be the configuration corresponding to a "yes" answer, and C_0 to "no". At this query step, N does the following:
CHOOSE(APPEND(ID_0(M',x'),C_1), APPEND(CHOOSE(ID_0(M',x'),**accept**),C_0).
In words, the number of accepting leaves of C_1 is multiplied by the number of accepting leaves of M' on x', whereas the number of accepting leaves of C_0 is multiplied by that of M' on x' *plus one.*

We now claim that N accepts x iff M does. To see this, notice that each leaf of the computation of M corresponds to a number of leaves in the computation of N which is the product of as many numbers as there are oracle steps in the computation leading to the leaf. If the path leading to the leaf corresponds to the correct answer to the query, then the multiplicant is odd, otherwise it is even. So, the leaf contributes an odd number to the total count (and thus it is taken into account as an accepting computation) iff it corresponds to correct answers to *all* the queries asked along the path. It follows that the total number of leaves of N is odd iff the total number of leaves of M, corresponding to correct sequences of oracle answers --that is, the leaves of M that are finally counted-- is odd. QED.

References

[An] D. Angluin "On counting problems and the polynomial-time hierarchy", *Theoretical Computer Science 12* (1980), pp. 161-173.

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

[Gi] J. Gill "Computational complexity of probabilistic Turing machines", *SIAM J. Computing, 6* (1977), pp. 675-695.

[LP] H.R. Lewis, C.H. Papadimitriou *Elements of the Theory of Computation*, Prentice-Hall, 1981.

[Pa] C.H. Papadimitriou "The complexity of unique solutions", *Proc. 13the FOCS Conference*, 1982, to appear.

[PY] C.H. Papadimitriou, M. Yannakakis "The complexity of facets (and some facets of complexity", *Proc. 14th STOC*, pp. 255-260, 1982. Also to appear in *JCSS*.

[Si] J. Simon "On the difference between one and many" *Proc. 4th Intern. Colloquium on Automata, Languages and Programming*, pp. 480-491, 1977.

[St] L.J. Stockmeyer "The polynomial-time hierarchy", *Theoretical Computer Science, 3* (1977), pp. 1-22.

[Va1] L.G. Valiant "The complexity of computing the permanent", *Theoretical Computer Science, 8* (1979), pp. 181-201.

[Va2] L.G. Valiant, private communication, August 1982.