ON THE DIFFERENCE BETWEEN ONE AND MANY

(preliminary version)

JANOS SIMON
DEPT. C. COMPUTAÇÃO
UNICAMP , CAMPINAS SP BRASIL

ABSTRACT

We examine the following question: 'Given a problem, is it more difficult to tell how many solutions the problem has than just deciding whether it has a solution?'. We show, that in specific cases, the question can be put into a mathematically meaningful form, namely when we can translate 'number of solutions' as 'number of distinct accepting computations of a nondeterministic Turing machine' (perhaps with appropriate weights). In this context, as we show, these questions are equivalent to problems about probabilistic machines (in the sense of Gill (9)).

In the first part of the paper we examine time-bounded computations, and justify our claim that this formalization is really the mathematical form of the question above by exhibiting a unifying model (the treshold machine) which has a special subcases the nondeterministic and the probabilistic machines. We show that natural complete problems exist and prove some elementary properties of the model.

In the second part we examine tape-bounded machines. We show that probabilistic tape-bounded machines may be simulated by deterministic Turing machines with only a polynomial increase in the amount of tape needed. This settles an open problem of Gill's (9).

This is a very powerful and perhaps unexpected result: it is the best known situation in wich we are able to show that powerful 'extras' like nondeterminism, get us only a polynomial improvement. The result is similar in content to Savitch's celebrated simulation of nondeterministic machines (20). The proof is completely unrelated to Savitch's (his construction does not work in the probabilistic case) and is quite involved, using some powerful recent results in complexity theory (10) (18) (4).

1.1. INTRODUCTION

A central question in the theory of computational complexity

may be formulated as: 'What is the difference (if any) between the difficulty of checking that a proposed solution to a problem is correct and the difficult of, given a problem, finding a solution to it?'. Of course, the question, as proposed, is meaningless, since we have not specified what we mean by 'problem' or by 'checking' 'finding' (i.e. we have not given a model of computation). Indeed, it is known, [2], [18], [10], that the answer depends crucially on the model of computation. Nevertheless, for the 'natural' model, the formalization of the question above becomes exactly the celebrated $P \stackrel{?}{=} NP$ question.

In this paper we shall try to formalize another question that is intuitively appealing: 'Is it more difficult to decide whether a problem has many (or a given number of) solutions than deciding whether it is solvable?'. Again, we shall have to give the question a precise mathematical meaning.

This in itself is not always necessarily possible: for example consider 'problem' as meaning 'problem in P'-i.e. problems to which the existence of solutions may be decided in time polynomial in the length of the imput by a deterministic. Turing machine (dTm). Then there are problems (where by a 'problem' we mean recognition of the language which contains the instances of the problem with a 'yes' answer - as in (12) p. 211). P1 and P2 such that

a) It is decidable, in polynomial time by a dTm whether an instance of P1 (P2) has a solution;

b) Given an integer k, and an instance x of P1, is decidable (by a dTm in time polynomial in $|k|$, $|x|$ whether x has k solutions, where $|x|$ denotes the length of x;

c) The similar problem for P2 - i.e. determining for an integer m and an instance y of P2 whether y has m solutions is NP-complete.

This follows by taking P1 to be 'given graph G, does it have a spanning tree?' and P2 to be 'given graph G with integer arc weights, does it have a negative cycle?', since there well known easy polynomial algorithms for finding negative cycles [14] and spanning trees [1]; the number of spanning trees of a graph may be computed using Kirchoff's classical matrix tree theorem [16], while it is not too hard to reduce (in the sense of Cook [3]) the problem of determining the number of negative cycles of a graph to the Hamiltonian cycle problem.

In the rest of the paper we shall show that there are con-

texts within which the question may be meaningfully formulated, and that it has interesting relationship to previously studied problems [9]. Finally we settle the problem for tape-bounded machines.

1.2. FORMULATION OF THE PROBLEM

We shall be very informal. Familiarity with standard terminology about NP-complete problems is assumed to be known (see [1],[11] for definitions) and definitions will be kept to a minimum. Proofs will only be outlined or omitted altogether.

DEFINITION The m-satisfiability problem is: 'Given a boolean expression B and an integer m, are there at least m different assignments of the values 'true' and 'false' to B's variables that result in B evaluating to 'true'?.

Generally, given a problem Q, we shall denote by m Q the problem 'does Q have at least m distinct solutions?'. Thus a nondeterministic Turing machine (ndTm) operating within bound b is said to m-accept the string x if there are at least m distinct computation sequences within bound b(x), that accept x. We shall call this kind of acceptance 'treshold acceptance with treshold m'. (The same ideas, within an algebraic context, have been presented independently in [5]).

DEFINITION Given two families of combinatorial problems A and B, a mapping f( ): A ——> B is *parsimonious* iff

1) Given a problem a $\in$ A, f(a) $\in$ B has exactly m solutions iff a has exactly m solutions.

2) f(a) may be computed in time polynomial in the size of a by a dTm.

Let A and B families of combinatorial problems and suppose there exist parsimonious mappings f( ): A ——> B and g( ): B ——> A. Consider the languages m A and m B (mA = {(ℓ,a) |a $\in$ A, a has at least ℓ distinct solutions}). Then

$$(\ell,a) \in mA \text{ iff } (\ell,f(a)) \in mB$$

and

$$(k,b) \in mB \text{ iff } (k,g(b)) \in mA$$

Thus, the recognition problems of the languages m A and m B have the same complexity (within a polynomial). Whenever such f( ) and

g( ) exist, we say that mA and mB are *related by parsimonious reductions*, and we use the notation f( ): mA ——> mB to denote also the function (k,a) ——> (k,f(a)).

DEFINITION mNP = {languages k-accepted within polynomial time by ndTm with some threshold k}.

The following theorem shows that for a huge number of combinatorial problems, the complexity of the question 'How many solutions does the problem have?' is the same (within a polynomial). We do not know what is the time required by a dTm or an ndTm to solve any of these problems, but we know that either they are simultaneously in P (NP) or all of then are outside of P (NP).

These sets are the complete sets (in the sense of Karp [15]) for the class of languages recognized within polynomial time by treshold acceptance by ndTm's. Definitions of the combinatorial problems may be found in the references.

THEOREM 1 The following combinatorial problems are related by parsimonious reductions:

1) m-SATISFIABILITY
2) Treshold acceptance with treshold m by polynomial time bounded ndTm's.
3) m-CLIQUE [15]
4) m-NODE COVER [15].
5) m-SET PACKING [15].
6) m-SET COVER [15]
7) m(0-1) INTEGER PROGRAMMING [15]
8) m-EXACT COVER [7]
9) m-HITTING SET [15]
10) m-KNAPSACK [15]
11) m-STEINER TREE [15]
12) m-JOB SEQUENCING [15]
13) m-PARTITION [15]
14) m-INEQUIVALENT REGULAR EXPRESSION OVER (+,o) [13]
15) m-MAX CUT [15]
16) m-MAXIMUM SATISFIABILITY with 2 LITERALS [8]
17) m-MINIMUM NODE NODE DELETION BIPARTITE SUBGRAPH [8]
18) m-NETWORK FLOW WITH MULTIPLIERS [19]
19) m-DIRECTED HAMILTONIAN CIRCUIT [15] , [23]
20) m-UNDIRECTED HAMILTONIAN CIRCUIT [15] , [23]

21) m-IMPOSSIBLE PAIRS CONSTRAINED PATHS [6]

The list is not exhaustive, it just purports to show that the class of NP-complete problems related by parsimonious reductions is quite extensive. In fact the only class of NP-complete problems that we tried but were unable to put in the list were the register allocation problems of Sethi [21], where to each assignment satisfying the boolean expression we have corresponding a constant number of programs, and this is a purely technical difficulty: it is possible to define an equivalence relation among the programs so that the relationship becomes 1:1.

The proof of the reductions are generally simply a check that the original reduction, which shows the problem to be NP-complete, is in fact parsimonious. In some cases, when this in not true, alternative parsimonious reductions exist in the literature [23], [7], or can be constructed.

Theorem 1 enables us to claim that the intuitive question we started with, may be formalized as

$$NP \stackrel{?}{=} mNP$$

Of course,

$$P \stackrel{?}{=} mNP$$

is another important problem. These problems seem very difficult.

1.3 PROPERTIES OF mNP

The treshold machine model can be visualized as follows: consider a ndTm M, and an input x. Build a *computation tree* of M on x the directed tree that has as a root the initial ID (instantaneous description) of M with input x, and where a node representing ID i has as sons nodes representing ID s j_i that are valid transitions of M from the configuration described by i. Without loss of generality, assume that M has at most 2 transitions from any given configuration and that M clocks itself (These restriction cause at most a polynomial increase of the running time). Thus the computation tree is a finite binary tree, where every path from the root to a leaf corresponds to a valid computation sequence of M on x and every valid sequence can be so obtained. The treshold machine accepts x iff the number of leaves corresponding to accepting ID's is greater than the treshold m. Using this description, it is easy to prove the following facts about the polynomial time bounded treshold languages mNP (languages recognized

by polynomial time bounded treshold machines).

1) mNP $\subseteq$ PTAPE
2) NP $\subseteq$ mNP
3) co-NP $\subseteq$ mNP

Gill defined probabilistic time bounded Tms as Tms that in some configurations could probabilistically choose between two equiprobable moves. A string is said to be accepted if there is a probability $p > 1/2$ that there will be an accepting computation on the string within the desired time bound [9].

THEOREM 2 mNP = {languages recognized in polynomial time by probabilistic Tms}.

It is clear that treshold machines can simulate probabilistic Tms. To prove the other direction we must show that the predicate 'more than half of the leaves accept' suffices to simulate any other treshold. This can be proven by padding the computation tree. In fact we can prove, using this techniques that the ' > 1/2' predicate can simulate the predicates $> \frac{p}{q}$, $= \frac{p}{q}$, $< \frac{p}{q}$ for positive integers p,q. (i.e. a fraction of more than, exactly, or less than p/q of the leaves must accept). In particular, we have the

COROLLARY mNP is closed under complement.

We do not know whether mNP is closed under union (or equivalently, under intersection), nor whether it is closed under existential polynomially bounded quantification (nondeterminism). One may define a hierarchy of alternating polynomially bounded quantifiers (as in the Stockmeyer - Meyer polynomial hierarchy [22], [17]) where existential and 'there exist at least half' quantifiers alternate. The ω-completion is again PTAPE, and again, not much else can be proven.

We now examine space-bounded probabilistic Tms.

2. SPACE-BOUNDED PROBABILISTIC Tms

Probabilistic Tms with bounded tape were also introduced by Gill [9]. Whereas it is known that nondeterministic and deterministic space bounded computations are polynomially related [20], the similar problem for probabilistic Tms was open. In this section we settle this problem. Our result, Theorem 3 states, that

PTAPE = probabilistic PTAPE

i.e., the use of probabilistic Tms enables us to have polynomial savings in tape, at most. This is not at all obvious: the straightforward simulation may run for indefinitely long time, Savitch's construction does not work for probabilistic Tm's, and the best result known was that

$$\text{probabilistic tape } [L(n)] \subseteq \text{deterministic tape } [2^{L(n)}].$$

The proof proceeds in a very roundeabout way, using heavily recent results about random access machines [10] [18] and fast algorithms [4]. It remains an interesting problem to give a direct simulation, or to justify the necessity of such a non-intuitive proof.

THEOREM 3 There is an integer k, such that for all tape-constructible functions $L(n) \geq \log n$, if a language is recognized within tape L(n) by a probabilistic Tm, it can be recognized within tape $(L(n))^k$ by a deterministic Turing machine.

COROLLARY probabilistic PTAPE = PTAPE

The proof of theorem 3 is through a series of lemmas, that show how to decide, in time polynomial in L(n), by an MRAM*, whether a given string x, $|x| = n$ is accepted by a probabilistic Tm M. Theorem 3 then follows from the results in [10] that show how such a computation can be simulated by a dTm.

LEMMA 1 Let $n = 2^t$. Two $n \times n$ matrices, the elements of which are integers of length at most n may be multiplied in time polynomial in t, by MRAM's, if the two matrices are initially stored in two registers of the MRAM.

PROOF All the machinery developed for and-or matrix multiplication in [18], [10] works for ordinary multiplication also, except that every element will be represented by a field of n bits. The shifts and the expanding and contracting operations require no modification, except to change maks of '$\emptyset$' and '1' by $\emptyset^n$ and 1^n. The problem is only to obtain in a single operation all the products. More precisely, the 'and-or' multiplication of $A = (a_{ij})$ by $B = (b_{ij})$ originally presented as

$$a_{oo}\ a_{ol} \cdots a_{on}\ a_{lo} \cdots a_{ln} \cdots a_{nn}$$

---

* MRAM - Random Access Machine with multiplication. See [10].

$$b_{oo}\ b_{o1}\cdots b_{on} \qquad\qquad b_{nn}$$

proceeds by reshuffling and duplicating the contents of the registers until they contain, for all i,j,k, a_{ij} and b_{jk} in corresponding positions. Then all products (and's) are computed in a single vector operation. Except for this step, the algorithm for integers and for bit are the same. To prove the lemma we have to show how to obtain efficiently the vector

$$C = c_o\ c_1 \cdots c_p$$

with $c_i = a_i b_i$, given

$$A = a_o\ a_1 \cdots a_p$$

$$B = b_o\ b_1 \cdots b_p$$

i.e. the componentwise product of A and B.

Let the a_i and b_j have length ℓ, and suppose that the a_i's are separated by ℓ 0's, i.e.

$$A_1 = a_o\ 0^{\ell}\ a_1 0^{\ell}.$$

Represent the elements of B as

$$B_1 = b_o\ 0^m\ b_1\ o^m \ldots$$

with $m = 2\ell p$. Then in a single multiplication and 0(log p) 'cleanup' operations we obtain C.

As it is generally the case with vector machines, the same algorithm can be used to multiply simultaneously k matrices, stored concatenated to each other.

We shall use lemma 1 to compute the probability of acceptance by probabilistic Tm's, by evaluating the limit of their probability matrices.

Let M be a probabilistic L(n)-tape-bounded Tm, x its input. Then it is possible to compute a $|\Sigma|^{L(|x|)} \times |\Sigma|^{L(|x|)}$ matrix P of transition probabilities among the Tm configurations. Without loss of generality, M will have exactly two transitions from each nonfinal configuration, all transitions will have a probability of 1/2, and there is a single accepting configuration. Then the elements of P will be 0 and 1/2 and there will be at most two nonzero elements per row.

Let $P_\infty = \sum_{k=0}^{\infty} P^k$

Then the probability of M accepting x is the element $(P_\infty)_{if}$ of P_∞, where i, f refer to the initial and final state respectively.

To compute (P_∞) if we first do as follows:

1) Given M and x compute 2P with the MRAM. (This can be done as in [10] [11]).

2) Delete from P all rows (substitute them for $\emptyset$ s) corresponding to configurations from which no sequence of transitions leads to the final configuration f. This can be done in polynomial time by computing the transitive closure of P.

Let Q' be the matrix thus obtained and $Q = \frac{1}{2} Q'$.

Then $(Q_\infty)_{if} = (P_\infty)_{if}$ and

$$Q_\infty = (I - Q)^{-1}$$

LEMMA 2 Let Q be an n x n stochastic matrix as above. The elements of Q_∞ are fractions of the form p/q, where $p,q < f(n)$

$$f(n) = 2^{cn \log n}$$

for some constant c, independent of Q.

PROOF Analysis of Csanky's first algorithm for evaluating inverses [4].

LEMMA 3 $(Q_\infty)_{if}$ may be computed in time $(\log n)^k$.

PROOF Csanky [4] proved that for an n x n matrix A, A^{-1} may be computed as

$$A^{-1} = \frac{n}{tr(AB_n)} B_n$$

$$B_n = (A - \frac{I}{n-1} Tr\ (A))(A - \frac{I}{n-2} tr\ A)...(A-tr(A))$$

This can be rewritten as

$$B_n = \frac{1}{2^n(n-1)!} [(n-1)C - Z][(n - 2)C - Z]...[C - Z]$$

where $C = 2A$, $Z = (2Tr\ A)I$

so that the elements of C and Z are integers. If A = (I - P),

Let $E = [(n-1)C - Z]\ldots[C - Z]$.

$$A^{-1} = \frac{n}{[2^n(n-1)!]^{-1}\ \mathrm{Tr}(AE)} \cdot \frac{1}{2^n(n-1)!}\ E = \frac{2nE}{\mathrm{Tr}(CE)} .$$

To compute E, matrix products may be evaluated using the algorithm of Lemma 1 and Lemma 2 (to prove that separators that are big enough can be inserted between elements). Subtraction may be done in parallel, and the products may be evaluated on logn parallel matrix multiplications. After we have E, we compute CE, using our matrix product, and sum the diagonal elements to obtain Tr(CE) (in $O(\log n^k)$ operations). Multiply E_{if} by 2n, this gives us the numerator while Tr(CE) is the denominator of the expression.

Now to decide whether M accepts x, compute both the numerator and the denominator of $(Q_\infty)_{if}$ multiply the numerator by 2 and accept if this is greater than the denominator.

This concludes the proof of the theorem.

To recapitulate, the simulation works as follows: given probabilistic Tm M and input x, get an MRAM that

1) computes the transition matrix, computes its transitive closure and eliminates useless entries

2) evaluates a series by computing an inverse, using Csanky's fast parallel inversion method and our programming tricks, sketched above.

3) Checks whether the probability of acceptance is greater than half.

4) This MRAM, in its turn is simulated by a deterministic Tm, using the results of [10].

All steps can be done at a polynomial loss of efficiency, proving the theorem.

As stated before, it is quite unsettling that a basic result like the above has such a tortuous proof. We were unable to supply a more direct proof.

CONCLUSIONS We have presented in the first part a model (the treshold machine) that unifies nicely in a single framework non-deterministic and probabilistic computations, proved some of their properties , but were unable to establish any truly nontrivial fact about them.

In the second part, we settled an important open problem about probabilistic space-bounded computations: that their space requirements are polynomially related to the tape used by deterministic computations. This fact is somewhat surprising, since probabilistic , models do seem more powerful. The proof is quite involved and uses in an essential manner, properties of MRAM's. It is an interesting question whether this is really necessary, and if so, why.

Acknowledgments I'd like to thank the long discussions and for the encouragement by prof. J. Hartmanis, and for fruitful conversations with Drs. J. Simon, Zvi Galil and M. Solomon. Financial support from FAPESP is gratefully acknowledgments (grant 75/1101).

BIBLIOGRAPHY

[1] Aho, A., J.E. Hopcroft and J.D. Ullman: The design and analysis of computer algorithms. Addison Wesley, Reading 1974.

[2] Baker, T., J. Gill and R. Solovay: Relativization of the P=?NP question. SIAM. J. COMP. 4:4, 431-442

[3] Cook, S.A.: The complexity of theorem-proving procedures. Proc. 3rd STOC (1971) 151-158,

[4] Csanky, L.: Fast parallel matrix inversion algorithms. SIAM J. COMP. 5:4, 618-623

[5] Eilenberg, S.: Automata, languages and programming Academic Press, N. York (1974) vol. A.

[6] Gabow, H. N., S. N. Maheshwari and L. Osterweil: On two problems in the generation of program test paths. IEEE Trans. Software Eng. 3: 2 (sept. 1976) 227-231.

[7] Galil, Z.: On direct encodings of nondeterministic Turing machines operating in polynomial time into P-complete problems. SIGACT News 6:1 (1974) 19-23.

[8] Garey, M. R., D. S. Johnson and L. Stockmeyer: Some simplified NP-complete problems. Proc. 6 th STOC (1974) 47-63.

[9] Gill, J. III: Computational complexity of probabilistic Turing machines. Proc. 6 th STOC (1974) 91-95.

[10] Hartmanis, J. and J. Simon: On the power of multiplication in random access machines. Proc. 15 th SWAT (1974) 13-23.

[11] Hartmanis, J. and J. Simon: On the structure of feasible computations in M. Rubinoff and M. C. Yovits (eds): *Advances in Computers* v. 14 Academic Press, N. York (1976) 1-43.

[12] Hoperoft, J. E. and J. D. Ullman: *Formal languages and their relation to automata.* Addison-Wesley, Reading Mass 1969.

[13] Hunt, H. B. III: On time and tape complexity of languages. Proc. 5 th STOC (1973) 10-19.

[14] Johnson, D.: Algorithms for shortest parths. TR 73-169 Cornell U.

[15] Karp, R. M.: Reducibility among combinatorial problems in R. E. Miller and J. W. Thatcher (eds): *Complexity of computer computations.* Plenum Press NY (1972) 85-104.

[16] Kirchoff, G.: Über die Auflösung der Gleichungen, auf welche man bei der Untersuchung der linearen Verteilung galvanischer Störme gefuhrt wird. Ann. Phys. Chem. 72 (1847) 497-508.

[17] Meyer, A. R. and L. J. Stockmeyer: The equivalence of regular expressions with squaring requires exponential space. Proc. 13 th SWAT (1972) 125-129.

[18] Pratt, V. R., L. Stockmeyer: A characterization of the power of vector machines. JCSS 12:2 (1976) 198-221.

[19] Sahni, S. L.: Some related problems from network flows, game theory and integer programming. Proc. 13 th SWAT (1972) 130-138.

[20] Savitch, W. L.: Relationships between nondeterministic an deter ministic tape complexities. JCSS 4:2 (1970) 177-192.

[21] Sethi, R.: Complete register allocation problems. SIAM. J. Comp. 4:3 (1975) 226-248.

[22] Stockmeyer, L. J.: The polynomial-time hierarchy. IBM Res. Rep. RC 5379.

[23] Valiant, L. G.: Unpublished manuscript.