MIT/LCS/TM-126

WITH WHAT FREQUENCY ARE APPARENTLY INTRACTABLE PROBLEMS DIFFICULT ?

Albert R. Meyer
Michael S. Paterson

February 1979

MIT/LCS/TM-126

WITH WHAT FREQUENCY ARE APPARENTLY INTRACTABLE PROBLEMS DIFFICULT?

A.R. Meyer and M.S. Paterson

February 1979

This report was prepared with the support of the National Science Foundation Grant No. MCS77-19754 A03.

Massachusetts Institute of Technology

Laboratory for Computer Science

Cambridge   Massachusetts 02139

WITH WHAT FREQUENCY ARE APPARENTLY INTRACTABLE PROBLEMS DIFFICULT?

by
Albert R. Meyer, M.I.T. and
Michael S. Paterson, Univ. of Warwick*

Abstract: An algorithm is almost polynomial-time (apt) iff there is a polynomial p such that for all n, the algorithm halts within p(n) steps on all but at most p(n) inputs of size at most n. It is shown that for **NP**-complete and polynomial space-complete problems, as well as certain other apparently intractable problems such as integer factoring, the following conditions are equivalent: (1) the problem is solvable by an apt algorithm, (2) the problem (or its complement) is polynomial-time transformable to a polynomial-sparse set, (3) the problem is solvable in polynomial time.

Five well-known decision problems which apparently cannot be solved by polynomial time algorithms[1] are:

(i) any polynomial space-complete problem,
(ii) any **NP**-complete problem,
(iii) graph isomorphism,
(iv) integer factoring,
(v) linear programming (i.e. deciding feasibility of a system of linear inequalities over the rationals).

Can these problems at least be solved by algorithms which are "fast" "nearly all the time"? In particular, can these problems be solved by algorithms which are *almost polynomial time* (*apt*)?

Definition 1. An algorithm is *apt* iff there is a polynomial p such that for all n, the algorithm halts within p(n) steps on all but at most p(n) inputs of size at most n. **APT** (**P**, **NP**, respectively) is the set of problems solvable by apt (polynomial time, nondeterministic polynomial time, respectively) algorithms.

*This work was supported in part by a National Science Foundation grant no. MCS77-19754 A03 and the M.I.T. Laboratory for Computer Science.

KEY WORDS: NP-complete, reducible, polynomial-time

Recently, P. Berman [2] has elegantly proved the theorem that there is an **NP**-complete problem solvable by an apt algorithm iff **P** = **NP** . In other words, if, as is widely supposed, **P** ≠ **NP**, then any algorithm solving an **NP**-complete problem must, for infinitely many n, take *more than polynomial time on more than polynomially many inputs* of size at most n.

In this note we present a simpler and more general version of Berman's result and apply this generalization to obtain among other corollaries the following:

Theorem 1. Let L be any of the problems (i) - (v) above. Then L is polynomial-time transformable (i.e. $\leq^P_m$) to a problem in **APT** iff L is in **P**.

Before proceeding, we should note that for L equal to any of the familiar examples of **NP**-complete or polynomial space-complete problems, it is obvious that L ∈ **APT** iff L ∈ **P**. The reason is that each of these familar examples allows one to "pad" any given problem instance into exponentially many different trivial variants of approximately the same size. For example, given an apt algorithm for recognizing satisfiable propositional formulas, one could decide in polynomial time whether an arbitrary formula F was satisfiable as follows: successively run the apt algorithm for a polynomial number of steps on the formulas F, $F \wedge x_1$, $F \wedge x_2$, ..., where x_i for $i \geq 1$ is a variable not appearing in F. After examining at most polynomially many such formulas, the apt algorithm must produce at least one response, and this response determines whether F is satisfiable, because F is satisfiable iff $F \wedge x_i$ is satisfiable.

On the other hand if the only reason for the frequent occurrence of hard instances of the satisfiability problem was the existence of the rather trivial kind of padding indicated above, then the possibility would remain that the satisfiability problem was polynomial transformable to a problem in **APT** -- because the padded instances could be transformed back to underlying "unpadded" instances. Theorem 1 rules out this latter possibility, unless **P** = **NP**. Thus the significant part of Theorem 1 is that it applies to any problem $\equiv^P_m$ to the examples (i) - (v). (Actually, it is not obvious that integer factoring, as formulated below, allows padding, so for L = integer factoring, even the statement L ∈ **P** iff L ∈ **APT** seems interesting.)

Note that every recursive subset of 0^* is in **APT**. An algorithm which rejects inputs not in 0^*, and on 0^* carries out any effective decision procedure for the set, no matter how inefficient, will be an apt algorithm. Thus Theorem 1 implies

Theorem 2. Let L be any of the examples (i) - (v) above. Then L is polynomial-time transformable to a subset of 0^* iff L ∈ **P**.

P. Berman observed that Theorem 1 implies Theorem 2, thereby settling a question raised by L. Berman and Hartmanis [1]. Actually, P. Berman introduced another technical condition which is (apparently) slightly weaker than polynomial-time transformability to a single letter alphabet set.

Definition 2. A language L is (polynomial) *sparse* iff there is a polynomial p such that for all $n \geq 0$ there are at most p(n) words of length at most n in L.

Let -L be the complement of L relative to Σ^*.

Definition 3. (P. Berman) A language $L \subset \Sigma^*$ is *sparse-reducible* iff there is a polynomial-time computable function $f{:}\Sigma^* \rightarrow \Delta^*$ such that $f(L) \cap f(-L) = \emptyset$ and $f(\Sigma^*)$ is sparse.

Lemma 1. Each of the following conditions on L implies its successor:

(a) $(\exists\, S \subset 0^*)\ [L \leq_m^p S]$,

(b) $(\exists\, S \in \mathbf{APT})\ [L \leq_m^p S]$,

(c) L is sparse-reducible,

(d) $(\exists\, S_1, S_2 \subset \Delta^*)$ [S_1 and S_2 are sparse sets, $L \leq_m^p S_1$, and $-L \leq_m^p S_2$],

(e) $(\exists\, S \subset \Delta^*)$ [S is sparse and $L \leq_m^p S$].

That (b) implies (c) was noted by P. Berman [2]. All the implications are easily proved.

P. Berman used condition (c), as the hypothesis for his result about **NP**-complete problems. S. Fortune [3] subsequently observed that Berman's argument using condition (c) could also be applied to polynomial space-complete problems.

In addition to generalizing the results to cover all the examples (i) - (v), we observe that condition (d), namely that both L and -L are polynomial-time transformable to sparse sets, suffices for the general theorem, and that condition (e) suffices for each of the examples (i) - (v) or else for their complements. (We believe that no two of conditions (a) - (e) are equivalent, but have not tried to verify this.)

In order to make our theorem easily applicable to (i) - (v) and similar examples, we introduce a technical definition which generalizes the partial order "is shorter than" on Σ^*.

Definition 4. Let $|x|$ denote the length of the string x. A partial order $<$ on Σ^* is polynomially well-founded and length-related, *OK* for short, iff

(a) there is a polynomial p such that every finite $<$-decreasing chain is shorter than p of the length of its maximum element, and

(b) $x < y$ implies $|x| \leq p(|y|)$ for some polynomial p, and all $x,y \in \Sigma^*$.

Obviously choosing to define $x < y$ iff $|x| < |y|$ is OK.

Definition 5. A language $L \subset \Sigma^*$ is *self-reducible* iff there is an oracle Turing machine, $\mathbf{M}$, and an OK partial order on Σ^* such that, given an oracle for L, $\mathbf{M}$ accepts L in polynomial time, and moreover, on any input $x \in \Sigma^*$, $\mathbf{M}$ asks its oracle only about words strictly less than x in the partial order.

For example, the set of satisfiable propositional formulas is self-reducible: reduce the problem of deciding F to problems of whether either of F_t and F_f are satisfiable, where F_t (F_f) is the result of setting the first variable in F to true (false) and simplifying; the OK order in this case is the one based on length.

The following problem, which is easily shown to be $\equiv_m^P$ to graph isomorphism, is also self-reducible: given two undirected finite graphs G_1, G_2 and a partial injection $f:V_1 \to V_2$, where V_i is the vertex set of G_i, can f be extended to an isomorphism between G_1 and G_2? The OK order we use is that $(H_1, H_2, g) < (G_1, G_2, f)$ iff $G_1 = H_1$, $G_2 = H_2$ and g properly extends f. The problem of whether (G_1, G_2, f) is acceptable reduces to at most $|V_2|$ "smaller" problems of the form (G_1, G_2, g) where g extends f by one domain element.

Similarly, let INTFAC = $\{ (m, a, b) \mid 1 < a \leq b \leq m$ and m has a prime factor between a and $b\}$. It is easy to see that INTFAC $\in \mathbf{P}$ iff there is an integer factoring algorithm which runs in polynomial time. Reduce the question whether $(m, a, b) \in$ INTFAC to the question whether $(m, a, c) \in$ INTFAC or $(m, c, b) \in$ INTAC, where $c = \lceil (a+b)/2 \rceil$. The OK partial order is:
$(m', a', b') < (m, a, b)$ iff $m = m'$, $a \leq a' \leq b' \leq b$, and $b' - a' \leq \lceil (b - a)/2 \rceil$.

The argument for self-reducibility of linear programming is similar. Thus we have:

Lemma 2. Let L be any of the examples (i) - (v) above. Then L is $\equiv_m^P$ to a problem which is self-reducible.

Main Theorem 3. If L and $-L$ are polynomial-time transformable to sparse sets, and L is self-reducible, then $L \in \mathbf{P}$.

Proof: To decide whether $x \in L$, consider the tree whose root is labelled with x, and such that every node labelled with any string y has, as its successive sons, nodes labelled $y_1, \ldots, y_k$ where $(y_1, \ldots, y_k)$ is the sequence of "smaller" strings about which the self-reduction asks its oracle given input y. This tree has only polynomial depth (by Def. 4a), all labels are of length polynomial in the length of x (by Def. 4b), and the degree of each node is bounded by a polynomial in the length of its label (by Def. 5). It can obviously be generated in order of a depth-first search in time bounded by a polynomial times the number of nodes.

The number of nodes however may be exponential. We therefore prune the tree as follows:

Let f be the polynomial time computable function transforming L to a sparse set; similarly for g and -L. Whenever all the sons of a node, say with label y, have been searched (and their membership in L consequently has been computed), then complete the self-reduction on y and let answer(y) be the truth value resulting from the self-reduction. Enter the pair (f(y), answer(y)) into a cumulative "answer list" of such pairs. Likewise enter (g(y), answer(y)) into a second such answer list. (In particular, some leaves of the tree correspond to reductions in which the oracle is not interrogated; they contribute the initial list entries.) When a node, say with label z, is first reached in the construction of the tree, abort the depth-first search below that node if there is already an entry for f(z) in the first list or for g(z) in the second list, using the answer in the entry for f(z) or g(z) as the answer for z.

It follows by definition of polynomial-time transformability and induction on the number of entries in the answer lists, that the answer for z obtained in this way is correct. Thus on completion of the construction of the pruned tree, the correct answer for whether $x \in L$ is obtained.

We claim the pruned tree has only polynomially many nodes, thereby completing the proof. To establish this last claim, observe that two *internal* nodes of the pruned tree (i.e. nodes with one or more unpruned sons) can have labels which map to the same value under f or g only if *they are on the same path from root to leaf* in the tree. Otherwise the depth-first search below one of them would have been completed, and entries made in the answer lists, before the other was first reached; but then the search at the node reached second would have been aborted before any sons were searched, contradicting the assumption that it was an internal node.

Now if we consider the internal nodes labelled with strings in L, we observe that there are only polynomially many distinct f values among them (since f(L) is sparse and |f(y)| is bounded by a polynomial in |y|) and each value occurs at most polynomially many times (since each path from root to leaf is no longer than a polynomial). So there are only polynomially many internal nodes labelled with words in L. A symmetric argument applies to internal nodes with labels not in L.

But, since the degree of all nodes is polynomially bounded, a polynomial bound on the number of internal nodes implies a polynomial bound on the total number of nodes. □

We remark that the preceding proof that the pruned tree has only polynomially many nodes should not be taken as completely obvious. Both f and g values may repeat many times, and there are potentially exponentially many distinct values.

Corollary 1: If some polynomial space-hard problem is polynomial-time transformable to a sparse set, then $\mathbf{P}$ = Polyspace.

Proof: Let Q be the set of closed quantified propositional formulas which are true. It is known that Q is complete in polynomial space [6, 7]. Moreover, Q is self-reducible since,

$\forall p F(p) \in Q$ iff $[F(\mathit{true}) \in Q$ and $F(\mathit{false}) \in Q]$,

$(F \wedge G) \in Q$ iff $[F \in Q$ and $G \in Q]$,

$\neg F \in Q$ iff $[F \notin Q]$.

Finally, $Q \equiv_m^P -Q$ since $F \in -Q$ iff $\neg F \in Q$.

So if L is polynomial space-hard, i.e. $Q \leq_m^P L$, and $L \leq_m^P$ to a sparse set, then (by transitivity of $\leq_m^P$) both Q and -Q are transformable to that sparse set. By Theorem 3, we conclude that $Q \in \mathbf{P}$ and hence $\mathbf{P}$ = Polyspace. □

S. Fortune [3] has observed for co-$\mathbf{NP}$-complete problems, in particular for propositional tautologies, that the hypothesis that TAUT is polynomial-time transformable to a sparse set suffices to imply TAUT $\in \mathbf{P}$. The reason is that in this case the self-reduction $F \in$ TAUT iff ($F_t \in$ TAUT and $F_f \in$ TAUT), is *conjunctive*.[2] In particular, if any node label not in TAUT turns up in the tree constructed in the proof of Theorem 3, the whole construction can be terminated because the root is not in TAUT.[3] Thus only the set f(TAUT) need be sparse.

We note that this observation applies equally well to (the complements of) problems (iii) - (v).

Theorem 4. If L is polynomial-time transformable to a sparse set and is conjunctively self-reducible, then $L \in \mathbf{P}$.

Corollary 2: Let L be any of problems (iii) - (v) above. If -L is polynomial-time transformable to a sparse set, then L ∈ **P**.

Proof: The complement of L is obviously conjunctively self-reducible using the self-reductions noted above. □

Actually for (v), linear programming, the Duality Theorem implies that the problem of deciding feasibility for a system of inequalities is $\equiv_m^P$ to the problem of infeasibility, so the reasoning of Corollary 1 could also have been used in this case.

A natural question left open by the preceding development concerns the class **W-APT** of *weakly* apt algorithms which, for some polynomial p and *infinitely many* n, halt within p(n) steps on all but at most p(n) inputs of size at most n.

1. Does Theorem 1 hold with "**W-APT**" replacing "**APT**"? We conjecture that it does.

The following technical problems also remain open:

2. For **NP**-complete problems, graph isomorphism, and integer factoring, does being polynomial-time transformable to a sparse set imply membership in **P**?

3. Are any two of the conditions given in Lemma 1 equivalent?

4. Is every problem which is polynomial-time transformable to a problem in **APT** necessarily itself in **APT**?

Another question which arises is whether the hypothesis of self-reducibility in Theorem 3 could be replaced by the simpler property of being in **NP**. This would be a significant improvement of Theorem 3 even if the sparseness condition (d) of Lemma 1 had to be strengthened to condition (a) or (b), for example. However, either such strengthening would be tantamount to solving one of the major open problems concerning nondeterministic computation, because using Theorem 3, the following assertions are easily shown to be equivalent

(i) there is a subset of 0^* in **NP** - **P**,
(ii) there is a subset of 0^* which is in **NP** and is not self-reducible,
(iii) nondeterministic exponentional time recognizability is not equivalent to deterministic exponential time recognizability.

Furthermore, using Theorem 3 and recent results of Landweber *et al.* [4], the following assertions are also easily shown to be equivalent:

(iv) there is a set in $(\mathbf{APT} \cap \mathbf{NP}) - \mathbf{P}$,
(v) there is a set in $\mathbf{APT} \cap \mathbf{NP}$ which is not self-reducible,
(vi) $\mathbf{P} \neq \mathbf{NP}$.

ACKNOWLEDGEMENTS: We would like to thank Zvi Galil for calling the work of P. Berman and S. Fortune to our attention, David Johnson for suggesting the application of Theorem 3 to linear programming and integer factoring, and Vaughan Pratt and Ronald Rivest for helpful remarks.

NOTES

1. For definitions of standard concepts such as polynomial-time algorithms, polynomial-time transformability (viz. $\leq_m^p$), etc., see any of the references.

2. Schnorr [5] makes interesting independent use of a notion he also calls "self-reducibility"; his notion amounts to a special case of conjunctive self-reducibility. (Schnorr's version is actually formulated disjunctively because he works with **NP** rather than co-**NP** problems.) The self-reducibility of the graph isomorphism problem was first observed by Schnorr.

3. The tree generating/pruning procedure may not, of course, immediately detect a label not in TAUT, so it continues until either a "false" entry is generated in the answer list of f values, or the polynomial bound on the number of f values is exceeded. Actually either of these two termination conditions suffices by itself to ensure that the constructed tree has only polynomially many nodes.

REFERENCES

[1] Berman, L. and Hartmanis, J. On isomorphisms and density of **NP**- and other complete sets. *SIAM J. Comp.* 6, 2 (June, 1977), 305-322.

[2] Berman, P. Relationship between density and deterministic complexity of **NP**-complete languages. *Fifth International Colloquium on Automata, Languages, and Programming*, Udine (July, 1978), *Springer Lecture Notes in Comp.Sci.*, 62, 63-71.

[3] Fortune, S. A note on sparse complete sets. Dept. of Computer Science,Cornell University, submitted for publication (Oct., 1978), 7pp.

[4] Landweber, L.H., Lipton, R.J. and Robertson, E.L. On the structure of sets in **NP** and other complexity classes, *Computer Sciences Technical Report* 342 (Dec., 1978), University of Wisconsin-Madison, 38pp.

[5] Schnorr, C.P. Optimal algorithms for self-reducible problems. *Third International Colloquium on Automata, Languages, and Programming*, Edinburgh (July, 1976).

[6] Stockmeyer, L.J. and Meyer, A.R. Word problems requiring exponential time: preliminary report. *5th Annual ACM Symposium on Theory of Computing* (1973), 1-9.

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

Cambridge, Mass
February, 1979