SPARSE COMPLETE SETS FOR NP: SOLUTION OF A CONJECTURE BY BERMAN AND HARTMANIS *

By

Stephen R. Mahaney

TR80-417

Department of Computer Science
Cornell University
Ithaca, New York 14853

*This research has been supported in part by National Science Foundation Grants MCS 75-09433 and MCS 78-00418.

Sparse Complete Sets for NP: Solution of a Conjecture by Berman and Hartmanis*

Stephen R. Mahaney

Computer Science Department
Cornell University
Ithaca, New York 14853

ABSTRACT

In this paper we show that if NP has a sparse complete set under many-one reductions, then P = NP. The result is extended to show that if NP is sparse reducible, then P = NP. The main techniques of this paper generalize the NP recognizer for the complement of a sparse complete set with census function to the case where the census function is not known (c.f. [HM]), then a many-one reduction of this language to the sparse set permits a polynomial time bounded tree search as in [B], [F], or [HP]. Even without actual knowledge of the census, the algorithm utilizes the properties of the true census to decide membership in SAT in polynomial time.

* This research has been supported in part by National Science Foundation Grants MCS 75-09433 and MCS 78-00418.

April 17, 1980

April 17, 1980

Sparse Complete Sets for NP: Solution of a Conjecture by Berman and Hartmanis

Stephen R. Mahaney

Computer Science Department
Cornell University
Ithaca, New York 14853

1. Introduction.

L. Berman and J. Hartmanis [BH] conjectured under the assumption $P \neq NP$ that all NP-complete sets are isomorphic; i.e. that there are polynomial time, bijective reductions with polynomial time inverse reductions between any two NP-complete sets. A consequence of this conjecture is that all NP-complete sets have equivalent density; in particular, no sparse set could be NP-complete unless P = NP.

P. Berman [B] gave a partial solution to this problem by showing that if a subset of an SLA language is NP-complete (a fortiori, sparse), then P = NP. This result was strengthened by Fortune [F] who showed that if co-NP has a sparse complete set, then P = NP. It was not necessary to assume that the satisfiable formulas reduce to a sparse set since the proof uses the conjunctive self-reducibility of non-satisfiable formulas and the sparse set to realize a polynomial time algorithm. Meyer and Patterson [MP] show similar results.

Hartmanis and Mahaney [HM] extended the results of Fortune and Meyer and Patterson by showing that if NP has a sparse complete set with an easily computable census function, then NP = co-NP; P = NP follows by Fortune's theorem. The question of how easy it is to compute census functions for NP-complete sets was left open.

In light of Fortune's observation about co-NP, the original conjecture by Berman and Hartmanis on reducing NP to a sparse set seems temptingly close. However the tree search methods of [B], [F], [MP] utilize the conjunctive self-reducibility of the co-NP-complete problem SAT^c.

In this paper we settle the conjecture by showing that if an NP-complete set is many-one reducible to a sparse set, then P = NP. Thus determining the existence of a sparse complete set for NP is equivalent to solving the P = NP? problem. We also show that the census function of a sparse NP-complete set is computable in P.

Section 2 contains definitions and an outline of the tree search method for showing that sparse sets for co-NP implies P = NP. Section 3 contains the main results; it assumes familiarity with the tree search methods.

2. Preliminaries.

We will consider languages over an alphabet Σ with two or more symbols. We assume familiarity with NP-complete sets (cf. [C], [K] or [AHU]). All the reductions in this paper will be polynomial time many-one reductions.

Definition: A subset S of Σ^* is sparse if there is a polynomial p(.) so that the number of strings in S of size at most n is at most p(n).

We restate the following theorem (cf. [F] or [HP]) and sketch the proof.

Theorem 1.1. If SAT^c is reducible to a sparse set, then P = NP.

Proof. Let f:SAT --> S be a reduction to a sparse set, S, and let F be a formula whose satisfiability is to be decided. We will search a binary tree formed from self-reductions of F as follows: F is at the root; a formula G with variables $X_1, \ldots, X_n$ occurring in the tree will have sons G_0 and G_1 where X_1 is replaced by false and true, respectively, and trivial simplifications are performed (e.g. true or X_i = true).

If the formula F has n variables, then the tree will have 2^n-1 nodes. We perform a depth-first search of this tree utilizing the sparse set to prune the search. At each node F' encountered we compute a label f(F'). We infer that certain labels correspond to SAT^c by the following:

i. When a node with formula false is found, its label is assigned "false."

ii. When two sons of a node have labels assigned "false," then the label of that node is also assigned "false." (This is the conjunctive self-reducibility of non-satisfiable formulas.)
We prune the search by stopping if a leaf has formula true in which case F is satisfiable by the assignments on the path to the leaf; and by not searching below a node whose label has already been assigned "false."

The following lemma establishes the polynomial running time of the algorithm.

Lemma 1.2. Let F be a formula with n variables. Let p(.) bound the density of S and let q(.) be a polynomial bounding the increases in size under the reduction f. Then the algorithm above visits at most

$$n + n * p(q(|F|))$$

interior nodes.

Proof. [F, HP]. Observe that if a label is expanded more than once, then the expansions are all on the same path from the root. Since path lengths are at most n+1 (with leaf), at most n * p(q(|F|)) interior nodes with label "false" are visited. A satisfying assignment visits at most another n nodes.

QED

Note that the algorithm does not require a sparse set of labels for satisfiable formulas. The sparse set of labels reduces the number of unsatisfiable formulas to be searched.

3. Solution of the Conjecture.

Initially, we establish the result for a sparse NP-complete set. The proof will be modified for the hypothesis that NP is sparse reducible.

The outline of the proof below is as follows: We first give an NP recognizer for a set similar to the complement of the sparse set S. Many-one reductions of this set to the sparse set are used to prove the existence of a sparse set of labels for SAT^c; however, the computation of this set of labels requires knowing the census of S. Finally, the depth-first search is modified to determine satisfiability of a formula

(without actually knowing the census value that will generate the sparse set of labels for SAT^c).

For the following discussion let $S \subset (0,1)^*$ be a sparse complete set for NP under many-one reductions. Let M_S be a non-deterministic polynomial time recognizer of S and let

$$c(n) = |S \cap (\Lambda+\Sigma)^n| \leq p(n)$$

where c(.) is the true census function of S, and p(.) is a polynomial that bounds the size of the census.

We begin by constructing a non-deterministic polynomial time Turing machine to recognize a "pseudo-complement" of S. The inputs include a padding, $\#^n$, and a potential census, k. Define the non-deterministic recognizer M by the following procedure:

$M(\#^n,s,k)$:

Check $|s| \leq n$; otherwise reject.

Check $k \leq p(n)$; otherwise reject.

Guess $s_1, \ldots, s_k$ so that

i. for all i, $|s_i| \leq n$.

ii. for all i and j, $s_i \neq s_j$.

iii. for all i, check that s_i is accepted by M_S.

iv. check that for all i $s \neq s_i$.

Lemma 3.1. Let $|s| \leq n$ and $k \leq p(n)$. Then on input $(\#^n,s,k)$ the machine M will:

a. accept if $k < c(n)$;

b. reject if $k > c(n)$; and

c. if $k = c(n)$, then M accepts if and only if M_S rejects s.

Proof of Lemma. We show part c. If M accepts, then it will have enumerated the elements of S up to size n, verified that they belong to S, and shown that s is distinct. Since k is the true census, M accepts if and only if s is not in S.

QED

Intuitively, for $k = c(n)$, M is a recognizer of S complement. Moreover, M accepts its language in non-deterministic polynomial time (the input $\#^n$ is a padding to ensure this).

We will require labelling functions for pruning tree searches. The following discussion shows how to construct such functions from the sparse set S and many-one reductions of L(M).

Since M is an NP machine and S is NP-complete, there is a P-time many-one reduction

$$g:L(M) \rightarrow S$$

so that for some monotonic polynomial q(.), inputs to M of size n are reduced to strings of size at most q(n) (cf. [C] and [K]). Similarly, for the NP-complete problem SAT, there is a P-time many-one reduction

$$f:SAT \rightarrow S$$

and a monotonic polynomial r(.) bounding the increase in size.

Let F of size m be a formula to be decided. Then any formula F' occurring in the tree of all self reductions will have size $\leq r(m)$. Regarding k as a possible value for c(n), we define

$$L_{n,k}(F') = g(\#^n, f(F'), k)$$

which will be the labelling function.

Lemma 3.2. Let F be a formula of size m. Let $n = r(m)$; i.e. n is a polynomial upper bound on the size of f(F') where $|F'| \leq m$. Finally, Let $k = c(n)$, the true census. Then the function

$$L_{n,k}(F')$$

for formulas F' of size at most m satisfies:

i. F' is not satisfiable if and only if L(F') is in S;

ii. The unsatisfiable formulas of size at most m are mapped by L to at most

$$p(q(2n+c'\log(n))) \leq p(q(3n))$$

distinct strings of S where c' is a constant depending only on p(.).

Proof: Part i. is immediate from Lemma 2.1. For part ii.,

observe that $2n + c' \log(n) \leq 3n$ is a bound on the size of $(\#^n, f(F'), k)$. Applying $p \circ q$ gives the census of strings in S that the triple could map onto.

QED

We now know that a suitable labelling function exists for $k = c(n)$; but we do not know $c(n)$! The algorithm in the following theorem shows how we can try $L_{n,k}$ for all $k \leq p(n)$.

Theorem 3.3. If NP has a sparse complete set, then P = NP.

Proof: We give a deterministic procedure to recognize SAT. Let F be a formula of size m. Apply the following algorithm:

```
begin
For  k = 0  to  p(r(m))  do
    Execute the depth first search algorithm with
        labelling function: L_{n,k} (F')
        at each node F' encountered in the pruned search tree.
    If a satisfying assignment is found,
        then halt; F is satisfiable.
    If a tree search visits more than
        n + n * p(q(3 r(m)))  internal nodes,
        then halt the search for this k.
    end;
F is not satisfiable;
end
```

The algorithm clearly runs in polynomial time since the loop is executed at most $p(r(m))$ times and each iteration of the loop visits at most a polynomial (in m) number of nodes.

The correctness of the algorithm is established in the following lemma:

Lemma 3.4. If F is satisfiable, then for $k = c(r(m))$ the search will find a satisfying assignment.

Proof: By Lemma 2.2. this k gives a labelling function that maps the unsatisfiable formulas of size at most m to a polynomially bounded set. Fortune shows that the depth first search will find a satisfying

assignment visiting at most

$$n + n * p(q(3n))$$

internal nodes.

QED

It is interesting to note here that we have not computed the census: a satisfying assignment could be found with any number of k's; similarly, if no satisfying assignment exists, many of the trees could be searched but the tree with k = c(r(n)) is not distinguished.

The method of conducting many tree searches is parallelled in the uniform algorithm technique by Karp and Lipton [KL]. They show that if NP could be accepted in P with log( ) advice, then P = NP. The census function might be compared to a log( )-advisor to the polynomial information in the set S.

It is not necessary to assume an NP recognizer for the sparse set: just that S is NP-hard.

Lemma 3.5. If S is sparse and NP-hard, then there is a set S# that is sparse, NP-complete, and has a P-time reduction: SAT --> S# that is length increasing.

Proof. Let f: SAT --> S be a P-time reduction and let # be a new symbol. Define f#: SAT --> S# by

$$f\#(F) = f(F)\#^{p}$$

where p = max{ 0, |f(F)| - |F| }. Clearly S# is sparse. The mapping f# reduces SAT to S#. Membership of s in S# is verified by guessing a satisfiable formula that maps to s and verifying satisfiability.

QED

Corollary 3.6. If NP is sparse reducible, then P = NP.

Lastly we remark that the census, c(n), of a sparse NP-complete set is computable in polynomial time. Indeed, assuming P = NP, the census of any sparse set in NP can be computed by standard techniques. If S is sparse and NP-complete, then P = NP by Theorem 3.3. so the census of S is computable in polynomial time. We have proved:

Corollary 3.7. If NP has a sparse complete set S, then the census of S is computable in P.

4. Discussion

Although the isomorphism results [BH] are the direct ancestry of the work discussed here, the concept of sparseness has another motivation. Can a "sparse amount of information" be used to solve NP problems in polynomial time? The approach here assumes the information is given as a many-one reduction to a sparse set.

For Turing reductions, the information is given as a sparse oracle. A. Meyer has shown that a sparse oracle for NP is equivalent to the existence of polynomial size circuits to solve NP (cf. [BH]). The recent work by Karp, Lipton and Sipser [KL] has shown that if NP has polynomial size circuits, then the polynomial time hierarchy collapses to Σ_2^P. Their result has a weaker hypothesis than Theorem 3.3. It is an important open problem to determine if polynomial size circuits for NP implies P = NP.

Acknowledgement.

I am greatly indebted to Juris Hartmanis and Vivian Sewelson for numerous discussions that lent insight into the methods developed in this paper. The uniform algorithm techniques of [KL] suggested the methed in Theorem 3.3. I am grateful to Richard Karp and Richard Lipton who circulated an early version of that paper.

References

[AHU] Aho, A., Hopcroft, J., and Ullman, J., The Design and Analysis of Computer Algorithms, Addison-Wesley (1974).

[BH] Berman, L. and Hartmanis, J., "On Isomorphisms and Density of NP and Other Complete Sets," SIAM J. Comput., 6 (1977) pp. 305-322. Also in Proceedings Eigth Annual ACM Symp. on Theory of Computing, (May 1976).

[B] 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.

[C] Cook, S. A., "The Complexity of Theorem Proving Procedures," Proc. 3rd Annual ACM Symposium on Theory of Computing, (1971) pp. 151-158.

[F] Fortune, S., "A Note on Sparse Complete Sets," SIAM J. Comput., 8 (1979), pp. 431-433.

[HM] Hartmanis, J. and Mahaney, S. R., "On Census Complexity and Sparseness of NP-Complete Sets," Cornell University Technical Report TR 80-416 (April 1980).

[K] Karp, R., "Reducibility Among Combinatorial Problems," in Complexity of Computer Computations (R. E. Miller and J. W. Thatcher, eds.), Plenum, New York (1972).

[KL] Karp, R. and Lipton, R., "Some Connections Between Nonuniform and Uniform Complexity Classes," Proc. 12th ACM Symp. on Theory of Computing, (May 1980).

[MP] Patterson, M. and Meyer, A., "With What Frequency are Apparently Intractable Problems Difficult?," M.I.T. Tech Report, Feb. 1979.