Math. Systems Theory 23, 209–225 (1990)

Mathematical Systems Theory

Power Indices and Easier Hard Problems*

R. E. Stearns and H. B. Hunt III

Department of Computer Science, State University of New York at Albany, Albany, NY 12222, USA

Abstract. The concepts of power_index, satisfiability hypothesis (**SH**), and structure tree are introduced and used to make sharper hypotheses about a problem's complexity than "the problem is *NP*-complete." These concepts are used to characterize the complexities of a number of basic *NP*-complete problems, including both CLIQUE and PARTITION which are shown to have power-indices at most $\frac{1}{2}$. Also, the problem 3SAT is shown to be solvable deterministically in time exponential only in the **square root** of $v + c$, where v is the number of variables and c is the number of "crossovers" needed to layout the formula in the plane.

1. Introduction

The concept of *NP*-completeness is of great practical importance because it enables us to make educated guesses about the deterministic time complexity of a numbr of important computational problems. If a problem is shown to be *NP*-complete, then we assume it takes exponential time and try to make the best of a bad situation. The same remark applies to *PSPACE*-complete and exponential-time-complete problems.

Knowing that a problem requires exponential time is an important but very crude understanding of a problem's complexity. Some problems requiring exponential time in fact require 2^n time, whereas others require only $2^{\sqrt{n}}$ time or even 2^{n^ε} time for some small ε. An $2^{O(n^{1/2})}$ algorithm might be considered close to practical (depending on constants of course) because $2^{n^{1/2}}$ operations for $n = 1000$ can be performed in under an our on a 1 mips machine. For $n = 31{,}991$, $2^{n^{1/3}}$ operations can be performed in an hour.

*The research of R. E. Stearns was supported by NSF Grants DCR 83-03932 and CCR 89-03319, and that of H. B. Hunt was supported by NSF Grants DCR 86-03184 and CCR 89-03319.

In this paper we present a methodology for making sharper hypotheses about a problem's complexity. This methodology enables us to say that some problem set A appears to "require about 2^n time" whereas some other problem set B only appears to "require about $2^{\sqrt{n}}$ time." We express this complexity using the concept of a "power index" introduced in Section 2. Roughly speaking, we say a problem has power index k if it can be solved in time $2^{\Omega(n^k)}$ and cannot be done any quicker. We do not know how to make unqualified assertions about the power index of a given NP-complete problem since we do not even know if $P \neq NP$. To overcome this difficulty, we make an assumption about the power index of SAT which we call the "satisfiability hypothesis" or "SH" for short. This hypothesis, introduced in Section 3, says the power index of SAT is one. This is a much stronger hypothesis than $P \neq NP$.

We want to prove statements such as "assuming SH, the power index of X is at least k" where k is as large as we can make it. The best k are obtained by using reductions of small size from problems which have large power index (assuming SH). In particular, this means it is important to understand the complexities of problems commonly used for reductions. We study two commonly used problems, namely CLIQUE and PARTITION in Sections 4 and 5. These are easy to analyze and provide simple examples of the methodology. Furthermore, it will be seen that (assuming SH) these problems have power index one-half. Thus, contrary to common practice, these problems should usually be avoided when making reductions.

In Section 6 we use power index tools to take a closer look at the difficulty of solving 3SAT. Using a concept we call a "structure tree," developed in greater detail in [SH3], we are able to solve instances of SAT much faster than by simple-minded exhaustive search. The complexity of our method depends exponentially only in the square root of $(v + c)$, where v is the number of variables and c is the number of "crossovers" needed to layout the 3CNF formula in the plane. In Section 7 we conclude the paper with some observations and open problems.

2. Power Indices

Following standard notation, we let $DTIME(T(n))$ be the set of problems that can be solved in deterministic time $T(n)$ on a multiple tape Turing machine. This enables us to define the power indices of problems and of functions $T(n)$ as follows:

Definition 2.1.

(1) Let A be a problem and let $I(A) = \{i \mid A \text{ is in DTIME}(2^{n^i})\}$. If $I(A) \neq \varnothing$, we define the *power index* of A (power_index(A)) to be the greatest lower bound of $I(A)$. If $I(A) = \varnothing$, we define power_index(A) to be ∞.

(2) Let $T: N \to N$. We define the *power index* of the function $T(\ )$ to be the greatest lower bound of the set $\{i \mid T(n) \leq 2^{n^i} \text{ for almost all } n\}$, if this latter set is nonempty, and to be ∞, otherwise.

Power indices have a certain robustness, which explains both their value and their limitations as is shown in the next easy proposition.

Proposition 2.2. *Let* $T: N \to N$.

(1) *If the function* $T(\ )$ *has power index* $r \geq 0$ *and* A *is in* $DTIME(T(n))$, *then* $power\_index(A) \leq r$.
(2) *If the function* $T(\ )$ *is increasing and has power index* $r \geq 0$, *then so do the functions* $(T(n))^s$ *and* $T(n \cdot (\log n)^s)$ *for all* $s > 0$.

The following are several direct consequences of Definition 2.1 and Proposition 2.2. First, for all problems A, power_index(A) is always defined since every nonempty set of nonnegative reals has a greatest lower bound. Second, every problem in **NP**, **PSPACE**, and **DEXPTIME** has a finite (possibly zero) power index. All problems A in **P** have power index zero. Third, power index is not sensitive to machine models whose runtimes are polynomially related to those of Turing machines. Fourth, power index is not sensitive to logarithmic factors in input representation.

For example, a Boolean formula that can be written with n symbols, if an infinite set of variable symbols is available, may require $n \log n$ symbols if encoded in binary. Thus the "try all assignments" algorithm for SAT would require $2^{\Theta(n)}$ time under the first measure and $2^{\Theta(n/\log(n))}$ time under the second. However by Proposition 2.2, the functions 2^n and $2^{n/\log n}$ have the same function power index (actually index one). This observation applies to any algorithm for SAT, and thus the power index of SAT is independent of which convention for measuring length is adopted.

Unlike NP-completeness, power index can be sensitive to polynomial variations in representation. For example, differing power indices may occur for a graph problem depending on whether the graphs are presented by incidence matrices or edge lists. This occurs because some incidence matrices have a size which is the square of the size of their edge list representations. Unless we specify otherwise, we assume that graphs are represented by edge lists. The edge-list notation is the one used in [K] and [GJ] to describe graphs. Because this notation is more compact, the power index under edge-list notation will always be at least as great as the power index under matrix notation.

Proposition 2.2 provides a method of establishing upper bounds on power index, namely finding an algorithm which is suitably fast. We now consider the establishment of lower bounds. One method of proving lower bounds involves finding "suitable" reductions. To understand what is meant by "suitable," we have to consider the *sizes* of the reductions.

Definition 2.3. A many-one reduction σ from problem A to problem B is said to be of *size* $s(n)$ if and only if, for almost all inputs x, $|\sigma(x)| \leq c \cdot s(|x|)$ for some constant $c > 0$.

The importance of size is seen by the next result:

Theorem 2.4. *If A has power index k and there is an $r > 0$ and d such that*

(1) *there is a size $n^r(\log n)^d$ reduction σ from A to B, and*
(2) *the deterministic-time complexity of σ is bounded by a function with function power index less than k,*

then the power index of B is at least k/r.

Proof. Assume that the power index of B is less than k/r. Then by (2) of Theorem 2.4 an $\varepsilon > 0$ exists such that the reduction can be performed in time $2^{n^{k-\varepsilon}}$ and by Definition 2.1 there is an algorithm for B running in time $n^{k/r-\varepsilon}$. An algorithm for A can then be obtained (in the usual way) as follows: given an x of size n to be tested for membership in A, apply the reduction to x and run the result $\sigma(x)$ through the algorithm for B. The time to do the reduction is $2^{n^{k-\varepsilon}}$. The time to test $\sigma(x)$ is $2^{|\sigma(x)|^{k/r-\varepsilon}} \leq 2^{n^{k-\varepsilon r}(\log n)^{d_1}}$ for some d_1. The sum of these times is a function with function power index less than k which contradicts the assumption about the power index of A. □

This theorem demonstrates the importance of size over time when considering reductions. The lower bound k/r on the power index of B depends directly on the size parameter r. However, the time of the reduction can be anything less than a certain exponential function without influencing the power index. Polynomial-time reductions are nice since they always satisfy (2) of Theorem 2.4 and thus they tell us something even when the actual power index of A is not known:

Corollary 2.5. *If there is a polynomial time and size $n^r(\log n)^d$ reduction from problem A to problem B, then power_index$(B) \geq$ power_index$(A)/r$.*

The $(\log n)^d$ factors in Theorem 2.5 and Corollary 2.6 imply that we can ignore logarithmic variations in representations when measuring the size of a reduction. We prefer to measure reduction sizes as if the inputs and outputs were represented with appropriate infinite alphabets for basic objects such as variables or nodes. This preference will not be formalized but will be following in many of our discussions.

3. The Satisfiability Hypothesis

Given a problem A with power index k_A, we can prove a problem B has power index k_B by

(1) providing an algorithm for B which has time complexity $T(n)$ where $T(n)$ has function power index k_B, and
(2) providing a reduction from A to B of size n^{k_A/k_B}.

Proposition 2.2 and Theorem 2.4 then establish k_B as the power index of B. This approach seems at first glance to be limited because we do not know the power index of any NP-complete problem (or even any $PSPACE$-complete problem). If any NP-complete problem can be shown to have nonzero power index, then all have nonzero power index and $P \neq NP$.

To overcome this problem, we make an educated guess about the power index of the satisfiability problem (SAT):

Satisfiability Hypothesis. power_index(SAT) = 1.

We abbreviate "Satisfiability Hypothesis" by **SH**. **SH** can be regarded as a strengthening of the "NP hypothesis" that "NP-complete problems take exponential time."

One evidence for **SH** is that no one has found an algorithm for SAT that runs in better than $2^{\Theta(n)}$ time (or $2^{\Theta(n/\log n)}$ if we wish to quibble about representations). However, there is other evidence as well. In [S3] it is shown that any problem in **NQL** (i.e., a problem that can be solved on a nondeterministic multitape Turing machine in time n polylog n or quasi-linear time) can be reduced to SAT in quasi-linear time (hence quasi-linear size). From Corollary 2.5 with $r = 1$, this means power_index(SAT) is at least as great as the power index of any language in **NQL**.

The heart of Schnorr's proof is an (n polylog n)-sized reduction of Turing machine computations into CNF formulas. (Cook's original construction [C1] was n^3 sized.) This implies that a method of solving SAT with power index less than one would give a method of doing nondeterministic simulations faster than backtracking. Other (n polylog n)-sized reductions are given in [R1] and [SH1]. See [C2] for more discussion on this.

There are linear size reductions from SAT to many of the common combinatorial problems. These include the following problems from Karp's original paper [K]: CLIQUE COVER, SATISFIABILITY, SET PACKING, NODE COVER, SET COVERING, FEEDBACK NODE SET, FEEDBACK ARC SET, DIRECTED HAMILTONIAN CIRCUIT, UNDIRECTED HAMILTONIAN CIRCUIT, at most 3SAT, CHROMATIC NUMBER, EXACT COVER, HITTING SET, STEINER TREE, 3-DIMENSIONAL MATCHING, and MAX CUT. From the six NP-complete problems called "basic problems" in [GJ], four make this list, namely 3-SATISFIABILITY (3SAT), 3-DIMENSIONAL MATCHING (3DM), VERTEX COVER (VC). and HAMILTONIAN CIRCUIT (HC). Linear reductions for these problems appear in many places in the literature and are not too difficult to find. A catalogue of linear reductions can be found in [D2] (see also [D1] and [SH1]). Theorem 2.4 tells us that, assuming **SH**, these problems have power index at least one and standard enumeration methods show this power index is at most one by Proposition 2.2. Thus, assuming **SH**, all the above problems have power index one. Following [S3], a problem A is called **NQL**-complete if A is in **NQL** and every problem in **NQL** is reducible to A by a quasi-linear time-bounded reduction. Quasi-linear time-bounded reductions of SAT are used in [S3] to prove that SAT and the 3-COLORABILITY and INDEPENDENT SET problems are **NQL**-complete. As one might expect, the

linear size reductions referred to above can be performed in quasi-linear time, and thus all of the problems on the above list are also **NQL**-complete. Even without assuming **SH**, Corollary 2.6 implies that all **NQL**-complete problems have the same power index.

At least two problems are conspicuously absent from the above list, namely CLIQUE and PARTITION. We say "conspicuous" because these are two of the "basic problems" given in [GJ]. We say "absent" because the best known reductions from SAT have size n^2. It turns out that these problems actually have algorithms (presented in the next two sections) that take $2^{O(\sqrt{n})}$ time. This enables us to assert the following:

Theorem 3.1. *Assuming* **SH**, *CLIQUE and PARTITION have power index one-half.*

Proof. The n^2-size reductions together with Corollary 2.5 imply power_index(CLIQUE) $\geq \frac{1}{2} \cdot$ power_index(SAT) and power_index(PARTITION) $\geq \frac{1}{2} \cdot$ power_index(SAT). Under **SH**, power_index(SAT) $= 1$ so power_index (PARTITION) $\geq \frac{1}{2}$ and power_index(CLIQUE) $\geq \frac{1}{2}$. The $2^{O(\sqrt{n})}$ algorithms to be given later together with Proposition 2.2 imply that one-half is also an upper bound on the power index of the two problems and the result is proven. □

In general, we hope to be able to find theorems of the form "Assuming **SH**, problem A has power index k." Theorem 3.1 is an example of such a theorem. Finding such results requires a two-pronged attack. The first prong is the somewhat esoteric search for efficient size-bounded reductions. The second is the practical exercise of trying to match our best reductions to algorithms of a corresponding complexity. When faced with a potentially exponential problem, we hope that power index analysis will lead us to easier exponential-time algorithms.

4. CLIQUE

The CLIQUE problem is this: given an integer in binary, a list of edges, and a list of nodes, determine if the corresponding graph has a clique of size k. Our results also apply if we take the word "CLIQUE" to mean the problem of finding the largest clique. Because CLIQUE is often used in *NP*-completeness proofs, it is important to understand the complexity of CLIQUE.

Given a graph with v nodes and e edges, there is an obvious $v^{O(\sqrt{e})}$ algorithm for solving CLIQUE, namely testing all subsets of nodes of size $\sqrt{e}$ or less. Since no clique can have more than $\sqrt{e}$ nodes, this testing will reveal all cliques. This algorithm is sufficient to prove Corollary 4.2, but is not as good as the following:

Theorem 4.1. *There is a* $2^{O(\sqrt{e})}$ *algorithm for CLIQUE, where e is the number of edges of the input graph.*

Proof. One such algorithm is given by the following: Assume an instance of (find the maximal) CLIQUE is given by a list L of e node pairs, each pair representing an edge of the graph. The following procedure finds the maximal clique in L.

Step 1. For each vertex v with edges in the graph, compute $E(v) = \{x | (v, x) \text{ in } L\}$ and pick the $\bar{v}$ which minimizes $|E(v)|$.

Step 2. Extract the subset

$$L_1(\bar{v}) = \{(x, y) | x \text{ in } E(\bar{v}) \text{ and } y \text{ in } E(\bar{v})\}$$

from L and apply this procedure to $L_1(\bar{v})$. Take the resulting maximal clique and add $\bar{v}$ to it to get the largest clique C_1 of L containing $\bar{v}$.

Step 3. Extract the subset

$$L_2(\bar{v}) = \{(x, y) | x \neq \bar{v} \text{ and } y \neq \bar{v}\}$$

from L and apply the procedure to $L_2(\bar{v})$ to get the largest clique C_2 of L not containing $\bar{v}$.

Step 4. Return the larger of C_1 and C_2.

This procedure obviously finds a maximal clique since it finds a maximal clique with node v, a maximal without v, and outputs the best result. In fact, the algorithm is not much more than a careful implementation of this simple plan. Only the simple heuristic for picking $\bar{v}$ distinguishes the procedure from an undirected search.

It remains to show that the above procedure for CLIQUE takes at most $2^{O(\sqrt{e})}$ deterministic time, where e is the number of edges of the input graph. The proof of this proceeds as follows: Let e be the number of edges ($e = |L|$) and let m be the number of nodes which have edges in L. We will prove by induction on e that the number of calls on the procedure is at most $m2^{\sqrt{2e}}$. Clearly, this is true when $m = 1$ since there is only one call.

Now assume the induction hypothesis is true for $m < \bar{m}$ and we will show it is true for $m = \bar{m}$. We consider two cases.

First consider the case where $|E(\bar{v})| < \sqrt{2e}$ where $\bar{v}$ is the vertex selected in the first step of the procedure. The calculation of C_1 takes less than $2^{\sqrt{2e}}$ calls since $L_1(\bar{v})$ has less than $|E(\bar{v})|$ vertices. By the induction hypothesis, the calculation of C_2 takes at most $(m-1)2^{\sqrt{2e}}$ calls since $L_2(\bar{v})$ does not have vertex $\bar{v}$. Altogether there are less than $(m-1)2^{\sqrt{2e}} + 2^{\sqrt{2e}} \leq m\, 2^{\sqrt{2e}}$ calls to compute C_1 and C_2 and thus at most $m\, 2^{\sqrt{2e}}$ for the entire procedure which is what we wanted to show.

Now consider the calls required when $|E(\bar{v})| \geq \sqrt{2e}$. Since $\bar{v}$ has the minimum $|E(v)|$, we have $\sum |E(v)| \geq m\sqrt{2e}$ where the sum is taken over the m vertices having edges in L. But, clearly, $\sum |E(v)| = 2e$ since the n edges of L have 2 endpoints and $\sum |E(v)|$ counts the total number of endpoints in L. Therefore $e \geq m\sqrt{2e}$ or $m \leq \sqrt{2e}$. Therefore in this case the procedure is only called at most $2^{\sqrt{2e}}$ times, which is less than $m\, 2^{\sqrt{2e}}$.

This concludes the proof that there are at most $m\,2^{\sqrt{2e}}$ calls. Since $m \leq n$ and the overhead of each call is some polynomial in e (depending on the machine model), the time of the procedure is at most $p(e)2^{\sqrt{2e}}$ for some polynomial p. This is $2^{O\sqrt{e}}$. □

Given the known $\Theta(n^2)$-size reductions from SAT to CLIQUE literature, we get the following immediate corollary of Theorem 4.1.

Corollary 4.2. *Assuming* **SH**, *CLIQUE has power index one-half.*

In general, we are interested in finding reductions which give the strongest possible conclusions about power index. Under Theorem 2.4, we have seen this means keeping the size of the reduction (the parameter r) small. But it also means picking hard problems A with large power indices k to reduce from. In particular, it is often better to reduce from a problem with power index one (such as SAT assuming SH) rather than with a problem with power index one-half (such as CLIQUE). Since we do not expect to find useful sublinear-sized reductions, the strongest conclusion we expect from reductions from CLIQUE is that a given problem has power index at least one-half.

The temptation to use CLIQUE in reductions has historically been strong. This goes all the way back to [K] where it occupied a prominent position in Karp's tree of reducibilities, and is perpetuated in Chapter 3 of [GJ] where it is called a "basic" problem. We now see CLIQUE should *not* be used unless we are sure the problem in question has power index at most one-half.

Conclusion 4.3. CLIQUE should be avoided as a problem for establishing *NP*-completeness.

CLIQUE is related to the problems INDEPENDENT SET and VERTEX COVER by means of "complement graphs." The *complement* of a graph (V, E) is the graph $(V, \bar{E})$ where $\bar{E} = V \times V - E$. Set $C \subset V$ is a clique in (V, E) if and only if C is an independent set in $(V, \bar{E})$ if and only if $V - C$ is a vertex cover of $(V, \bar{E})$. This relationship prompted Garey and Johnson to call these problems "really just different ways of looking at the same problem" [GJ, p. 53]. This statement is appropriate as far as *NP*-hardness is concerned but breaks down when power indices are considered or when solving by exhaustive search.

If graphs are represented by edge lists, the complement of a graph with v nodes and zero edges will be a graph with v nodes and v^2 edges. Thus converting a graph to it complement is a size n^2 operation. If, on the other hand, graphs are represented by incidence matrices, then a graph and its complement are represented by matrices of equal size and taking a complement is a linear size operation. Putting together these facts gives the following:

Theorem 4.4. *Assuming* **SH**:

(1) *VERTEX COVER and INDEPENDENT SET have power index one using edge-list representations and power index one-half using incidence-matrix representations.*
(2) *CLIQUE has power index one-half under either representation.*

Proof. All three problems must have the same power index using incidence-matrix representations since the problems are linearly reducible to each other. This index is at most one-half since the number of vertices is $\sqrt{n}$ where n is the number of matrix entries, so there are at most $2^{\sqrt{n}}$ subsets of vertices to be considered when solving by exhaustive search.

VERTEX COVER and INDEPENDENT SET using edge lists have power index one because of known linear reductions from SAT. Converting from edge lists to incidence matrices is itself a reduction of size n^2 so the power indices of the matrix representations must (assuming **SH**) be at least one-half (Corollary 2.6). □

Statement (1) of Theorem 4.4 shows that switching to a less-efficient representation can, as expected, reduce the power index. Statement (2) of Theorem 4.4 shows, however, that this need not always be the case. The intuitive reason CLIQUE remains the same is that the most difficult incidences of the problem have many edges and have about equal size under both representations.

5. PARTITION

The PARTITION problem is this: given a set of numbers in binary notation, determine if the numbers can be partitioned into two sets which have the same sum. Our results apply equally well to the problem of finding such a partition. The size of the problem is taken to be the total number of bits required to represent the numbers. Two well-known algorithms to solve this problem are enumeration and dynamic programming as discussed in [GJ]. Both methods take about 2^n time. We blend these two techniques to get a $2^{O(\sqrt{n})}$ algorithm.

Assume now that a set N of numbers in nonbinary notation is given so that n bits are used to represent all the set N. The following procedure determines if N can be partitioned into equal size sets:

Step 1. Partition N into two sets A and B such that $a \geq b$ for all a in A and b in B and such that the following expression is minimized:

$$\max\left(2^{|A|}, \sum_{i \in B} i\right).$$

Step 2. Let $S = \sum_{i \in B} i$. Create an array Z of the size S and assign $Z[i]$ value 1 if and only if some subset of B adds up to i. This is done using the dynamic-programming method.

Step 3. Consider each partition of A into two sets and let T be the sum of the numbers in one block minus the sum of the numbers in the other block. If $|T| > S$, go on to the next partition of A to be considered (if any else reject). If $|T| \leq S$, then accept if B can be divided so that the difference between the sums is also $|T|$. This happens exactly when $S - |T|$ is even and $Z[(S - |T|)/2] = 1$.

With a little extra bookkeeping, the above algorithm can be used to exhibit a partition if one exists.

Theorem 5.1. *The above procedure for PARTITION takes at most* $2^{O(\sqrt{n})}$ *time where n is the total number of bits used to represent the numbers.*

Proof. Step 1 is clearly polynomial. One of the partitions into A and B has the property that all numbers with $\sqrt{n}$ bits or more belong to A and all others belong to B. Under this partition, $2^{|A|} \leq 2^{\sqrt{n}}$ because there can only be $\sqrt{n}$ numbers of length $\sqrt{n}$ in the input. Also, $S = \sum_{i \in B} i < n2^{\sqrt{n}}$ since B has at most n numbers and each has length less than $\sqrt{n}$ bits. The max of the two numbers is thus less than $n2^{\sqrt{n}}$.

The dynamic programming of step 2 can be done with $|B|$ scans of the array Z. Since $|B| \leq n$ and $S < n2^{\sqrt{n}}$, this takes at most $n^2 2^{\sqrt{n}}$ time.

The number of partitions of A in step 3 is $2^{|A|}$ which is less than $n2^{\sqrt{n}}$ because of step 1. The time taken to look up a value in Z varies with computer model and could take from one time unit on a RAM to $n2^{\sqrt{n}}$ units on a Turing machine. Thus the time of step 3 could be from $n2^{\sqrt{n}}$ to $n^2 2^{\sqrt{n}}$. This is $2^{O(\sqrt{n})}$ in either case.

Since each step can be performed in $2^{O(\sqrt{n})}$, so can the entire procedure. □

Corollary 5.2. *Assuming SH, the power index of PARTITION is one-half.*

Proof. There are known n^2 reductions from SAT to PARTITION so the power index (assuming **SH**) of PARTITION is at least one-half. Theorem 5.1 shows it is at most one-half. □

As with CLIQUE, reductions from PARTITION can only bound power indices below by one-half and should be avoided if a stronger result is desired. However, we believe many problems that resemble PARTITION are likely to have power index one-half (assuming **SH**). For example, this is true of KNAPSACK and SEQUENCING from [K]. (Proofs of this assertion are in [SH1].) Thus we do not make a strong recommendation against using PARTITION.

The algorithm given above uses exponential space as well as exponential time because step 2 uses an array of exponential size. The algorithm can be altered into a linear-space algorithm at considerable cost in running time. The algorithm is as above except that the array Z is not created. At the point in step 3 when the predicate $Z[(s-|T|)/2] = 1$ is to be tested, a "divide and conquer" recursive subroutine is called to see if a subset of B adds up to $(S-|T|)/2$. This subroutine divides B into equal-sized parts and seeks a number h such that a subset of the first

half sums to h and a subset of the second half sums to $(S-|T|)/2-h$. The subroutine must remember $\log n$ numbers of length $\sqrt{n}$ and may require time $n^{O(\sqrt{n})}$. With this change, the overall algorithm runs in time $n^{O(\sqrt{n})}$. Although this time function also has function power index one-half, it is significantly more time than the original algorithm.

Robson provides additional insights into PARTITION in [R2], particularly in regard to simultaneous $2^{O(\sqrt{n})}$ time and linear space. The main ideas of the PARTITION algorithm can also be found in technical report [S1].

6. SAT Revisited

In this section we consider a method of organizing the variables and clauses of a SAT instance into a "structure tree" so that an exhaustive search for a solution can be performed more efficiently. Our immediate objective is to prove a result about the complexity of SAT instances, namely that SAT can be solved in time $|F| \cdot v^{O(\sqrt{v+c})}$ where $|F|$ is the formula size, v is the number of variables, and c is the number of "crossovers" needed to lay out the instances on the plane. This means that if SAT does have power index one, the hardest instances of SAT all have the property that they cannot be laid out in the plane without having $\Theta(n^2)$ crossovers. This suggests, for example, that the problems that arise naturally from the analysis of electronic circuits may all be substantially easier than the general case.

A second reason for discussing trees here is that this approach is widely applicable to large clauses of problems, including all problems of a finite algebraic nature, and this enables us to understand the complexity of broad classes of problems without reviewing each NP-complete problem individually. (We prove this in [SH3].)

We consider a CNF formula F to consist of two parts, a set of Boolean variables V and a set of clauses P. We use the letter "P" for the clause set rather than "C" because we usually think of the clauses as "predicates." We define the size of $F=(V, P)$ written as $|F|$, to be $\sum_{p \in P} |p|$ where $|p|$ is the number of variable instances in clause p. V_p is the set of variables in p.

Definition 6.1. Give a CNF formula $F=(V, P)$, define a *structure tree* S for F to be an ordered triple (T, α, β) where

(1) T is a rooted tree with node set N,
(2) α: $V \to N$ is called the *variable association*,
(3) β: $P \to N$ is called the *predicate association*, and
(4) for all y in V and p in P, y in V_p implies $\alpha(y)$ is an ancestor of $\beta(p)$ in T.

For n in N, define $A(n)=\{y \text{ in } V | \alpha(y)=n\}$ and $B(n)=\{p \text{ in } P | \beta(p)=n\}$. We call these the set of variables and the set of predicates *associated* with tree node n.

This definition is somewhat redundant in that a suitable predicate association β can be computed from the variable association α and vice versa. (Define $\beta(p)$ to

be the lowest node $\alpha(v)$ for any variable v in p, or define $\alpha(v) = \text{lub}\{\beta(p) | v \text{ in } p\}$.) However, we find it helps our thinking to give these concepts equal status.

Two parameters of a structure tree useful in complexity are "weighted depth" and "channelwidth" defined as follows:

Definition 6.2. Given a CNF formula $F = (V, P)$ and a structure tree $S = (T, \alpha, \beta)$ for F, we say that:

(1) y in V is a *branch variable* at node n of T if and only if $\alpha(y)$ is an ancestor of n.
(2) y in V is a *channel variable* of node n of T if and only if it is a branch variable and there is a p in P such that y in V_p and $\beta(p)$ is a descendant of n in T.

The set of branch variables of n is called $\text{BV}(n)$ and the set of channel variables is called $\text{CV}(n)$. Define the *weighted depth* of S by $\text{WD}(S) = \max\{|\text{BV}(n)| | n \text{ in } T\}$ and the *channelwidth of* S by $\text{CW}(S) = \max\{|\text{CV}(n)| | n \text{ in } T\}$.

Given a structure tree S for F, there is a fairly straightforward backtracking algorithm for testing the satisfiability of F in time $O(|F| \cdot 2^{\text{WD}(S)})$. The algorithm works on each node t of the structure tree by assigning all possible values to variables in $A(t)$, testing the assignment in clauses in $B(t)$, and calling itself recursively when the tests succeed. The point is that the children of t can be tested independently since they are associated with disjoint variable sets. If the same procedure is run with tables of partial results to prevent recomputing a result, the procedure can be done in time $O(|F| \cdot 2^{\text{CW}(S)})$. A third approach using tables in the bottom-up style of dynamic programming also runs in time $2^{O(\text{CW}(S))}$. The methods based on channelwidth may seem superior because $\text{CW}(S) \leq \text{WD}(S)$ for all structure trees S, but the weighted-depth-based method uses essentially linear space whereas the others use exponential space.

The above discussion implies that knowing a good structure tree, namely one of small weighted depth, enables us to test satisfiability much faster than in an unorganized exhaustive search. Finding good structure trees becomes a useful and powerful tool for identifying "easier hard problems" or "easier" problem instances. In the general case, we can fall back upon the following:

Lemma 6.3. *Given a CNF formula $F = (V, P)$ and an integer D, there is an $O(|F| \cdot |V|^D)$-time algorithm to determine if F has a tree of weighted depth D or less and to produce such a tree if one exists.*

We do not have sufficient space here to prove this result. The algorithm is based on exhaustive searches for sufficiently small sets of variables which act as hypergraph or subhypergraph separators where the clauses are represented by hyperedges. At any node n of a structure tree, $\text{BV}(n)$ and $\text{CV}(n)$ are separators.

From a power-index point of view, the algorithm of Lemma 6.3 is fast enough in that, for $D = |F|^k$, we can spend $O(|F| \cdot |V|^D)$ time finding a structure tree and then spend $O(|F| \cdot 2^D)$ time with the tree testing satisfiability. The two steps combined have a time bound with function power index k.

We now look at one application, namely the relationship between the complexity of SAT and the existance of good planar layouts. We confine our discussion to 3SAT. First consider the problem of PLANAR 3SAT. As defined in [L], a CNF formula is planar if a certain underlying bipartite graph is planar. This graph has a node for each variable, a node for each clause, and edges between each clause node and the nodes for the clause's variables. It is shown in [L] that this problem is *NP*-complete. The reduction given is size n^2, which leaves open the possibility that the power index of planar 3SAT is only one-half. The power index is in fact one-half (assuming as usual **SH**) because there is a $2^{O(\sqrt{n})}$ algorithm for solving this problem [RH].

The algorithm is based on the idea of first finding a structure tree of weighted-depth $O(\sqrt{n})$ and then using the algorithm already discussed. To find the tree, lay the graph out in the plane using the linear algorithm in [HT] and then make recursive application of the planes separator theorem from [LT]. These separator techniques must be adjusted somewhat since we need variable sets which separate rather than node sets.

Now consider the graph of a 3CNF or formula F drawn in the plane with "crossover points" where two edges are drawn crossing each other. By treating the crossover points as nodes of a graph, we can construct a separator tree for the formula by recursive application of the planar separator theorem to the graph. As a consequence, given a layout with c crossovers, a structure tree with weighted depth $O(\sqrt{v+c})$ can be constructed in polynomial time and then satisfiability tested in $|F| \cdot 2^{O(\sqrt{v+c})}$ time.

Now consider the situation where we have a 3CNF formula which has a layout with c crossovers but we do not know the layout. The very existence of the layout implies the existence of a structure tree with weighted depth $O(\sqrt{v+c})$, and we can search for such a tree directly using Lemma 6.3 without actually constructing the layout. This gives us the following:

Theorem 6.4. *A 3CNF formula F can be tested for satisfiability in time* $|F| \cdot v^{O(\sqrt{v+c})}$ *where v is the number of variables in F and c is the minimum number of crossovers needed to layout F in the plane.*

This theorem gives us an additional parameter, namely the crossover number, with which to analyze complexity. There we see that the power index of 3SAT is only affected by the square root of the number of variables. If **SH** is true, then the difficult instances are all formulas which require $\Omega(v^2)$ crossovers no matter how they are laid out.

The crossover parameter is nice in that it is "continuous" in the sense that small changes in a formula only cause small changes in the crossover parameter. This is in contrast with "binary" parameters such as "planarity." It may only take one addition of an edge to change a planar graph into a nonplanar graph. Continuous parameters give insight into the complexity of problem instances whereas binary parameters address only the complexity of special cases. It is probably rare that one deals with a formula that is strictly planar, but it may be common to work with formulas which have $O(v)$ crossovers or less. Such

"near planar" formulas can also be solved in $|f| \cdot v^{O(\sqrt{v})}$ time, much faster than the assumed worst case, and are probably the normal case encountered when analyzing digital circuits laid out on a chip.

The structure-tree-based algorithms are easily modified to find solutions, count solutions, test uniqueness, or find best assignments as measured by the number of clauses satisfied. The structure tree can be used to explain and generalize the work of many authors. This includes our own work [SH1], [HS1], [HS2], [RH], [SH2], and [HRS]; the work of Lipton and Tarjan [LT] and others on planar graphs; the work of Rosenthal on nonserial dynamic programming [R3]; Schaefer's work on generalized satisfiability problems [S2]; and the work of Robertson and Seymour [RS], Bodlaender [B], Arnborg, *et al.* [ALS], and others on graphs of constantly bounded treewidth. A very general exposition is given in [SH3]. The structure tree concepts apply very generally, not just to *NP*-complete problems, and many of the above references deal primarily with polynomial-time algorithms.

7. Observations and Open Problems

We have seen that the power index concept and the Satisfiability Hypothesis enable us to make complexity distinctions among various *NP*-complete problems and potentially among *PSPACE* and *EXPTIME*-complete problems. This approach also provides insights as to *why* problem instances are hard. For example, we have seen that the number of crossovers is an essential ingredient in making instances hard.

We hope that the complexity of many problems can be understood from general principles rather than be settled piecemeal on a case-by-case basis. One such approach is to identify classes of problems for which good structure trees can be obtained. For certain graph problems, good structure trees are inherited directly from the graph, and the structure tree can be used to explain algorithms already in the literature. This includes certain problems for planar graphs [LT] which have $2^{O(\sqrt{n})}$ or $n^{O(\sqrt{n})}$ algorithms and for graphs of bounded bandwidth [MS] and bounded treewidth [RS] which have polynomial algorithms. However, the structure tree is really an object for controlling a search rather than for describing problems, and any method of finding good separators, small weighted depths, or small channelwidth give good algorithms.

The approach of "looking for good structure trees" has an additional advantage in that small changes in a problem may only cause small changes in the structure tree and hence small changes in complexity. It thus becomes useful to have concepts which change gradually with changes in problem instance. For example, counting crossovers becomes important and displaces the cruder concept of planarity.

We consider the complexity of an *NP*-hard problem as "understood" if we can establish a lower bound on its power index via efficient reductions and **SH**, and establish an upper bound via some algorithm. We have been able to do this in many cases, often quite easily. However, some cases have arisen where we have

been unable (so far) to characterize their power index and where new insights are needed. We mention four of these here.

First consider 3-PARTITION discussed extensively in Chapter 4 of [GJ]. This is shown *NP*-hard first by a linear reduction of 3D-MATCHING to 4-PARTITION and then a square-size reduction from 4-PARTITION to 3-PARTITION. Since 3D-MATCHING is as hard as SAT, the linear reduction establishes 4-PARTITION as having the same complexity (which we believe, under **SH**, is power index one). Because of the square-size reduction to 3-PARTITION, that problem might have a $2^{O(\sqrt{n})}$ deterministic-time algorithm. Yet we have nothing better than 2^n so the best we know under **SH** is that 3-PARTITION has a power index between one-half and one. Until this matter is resolved, it is better to use reductions from 4-PARTITION. For example, [GJ] gives a linear reduction to SEQUENCING WITHIN INTERVALS where the main idea works equally well from either 3-PARTITION or 4-PARTITION. The mapping from 4-PARTITION is the more significant since it proves SEQUENCING WITHIN INTERVALS has the same power index as SAT; a strictly stronger conclusion than can be made now from 3-PARTITION.

Next consider the problem of dividing the nodes of a graph into two equal-sized sets so that the number of edges between subsets is minimized. (This is a special case of MINIMUM CUT INTO BOUNDED SETS.) The reduction is obtained from a simple version of MAX CUT by taking the complementary graph. This reduction is square-size but maps the hard MAX CUT problems into easier instances of MINIMUM CUT (easier because trying all cuts is $2^{\sqrt{n}}$ when the number of edges is $O(|V|^2)$). However, there is no reason to believe there is an $O(2^{\sqrt{n}})$ algorithm which includes problem instances where the number of edges is $O(|V|)$.

The third specific problem we want to mention is STAR FREE REGULAR EXPRESSION INEQUIVALENCE [H], [SM]. The reduction from [H] is square-size. This mapping goes into a very special case, namely expressions where each state of the corresponding nondeterministic finite-state automaton can only be reached by strings of one particular length. We know how to solve this special case in time $2^{O(\sqrt{n})}$ but do not know how to solve the general case in less than $2^{O(n)}$.

The final problem is GEOMETRIC TRAVELING SALESMAN (given points in the plane, find a tour with minimum discretized Euclidean distance). This has been shown to be *NP*-complete [P], [GGJ], [JP]. Following [JP], this is proven via reductions from planar HC to grid graph HC and then grid graph to "EUCLIDEAN TSP DECISION" which is the GEOMETRIC TSP of [GJ]. But planar HC already has power index less than or equal to one-half, as we have shown in [SH1], and the reduction to grid graphs is itself size n^2. Thus, given our current state of knowledge, GEOMETRIC TSP could have power index as low as one-quarter and could thus be solvable in many practical instances. We understand that Warren Smith has an $n^{O(\sqrt{n})}$ algortihm for this problem in his Ph.D. thesis.

We also note that paying attention to reduction sizes can also be useful in studying *P*. Analogous to Definition 2.1, we can define the polynomial index of *A* to be the greatest lower bound on the set of *k* such that *A* is in $DTIME(n^k)$. Theorem

2.4 and Corollary 2.5 go through with the words "polynomial index" substituted for "power index." This makes it possible to make statements about the relative complexities of NP-complete problems which hold even if $P = NP$. Thus for example, NQL-complete problems also have the same polynomial index.

References

[ALS] S. Arnborg, J. Lagergren, and D. Seese, Which problems are easy for tree-decomposable graphs?, unpublished manuscript, 1987.

[B] H. L. Bodlaender, Dynamic programming on graphs with bounded tree width, Technical Report RUU-CS-87-22, Dept. of Computer Science, University of Utrecht, Utrecht, The Netherlands, 1987.

[C1] S. A. Cook, The complexity of theorem-proving procedures, *Proc. Third Annl. ACM Symp. on Theory of Computing*, pp. 151-158, 1971.

[C2] S. A. Cook, Short propositional formulas represent nondeterministic computations, *Inform. Process. Lett.*, vol. 26, pp. 269-270, 1988.

[D1] A. K. Dewdney, Linear time transformations between combinatorial problems, *Internat. J. Comput. Math.*, vol. 11, pp. 91-110, 1982.

[D2] A. K. Dewdney, Generic reduction computers, Report No. 141, Dept. of Computer Science, University of Western Ontario, 1985.

[GGJ] M. R. Garey, R. L. Graham, and D. S. Johnson, Some NP-complete geometric problems, *Proc. 8th Ann. ACM Symp. on Theory of Computing*, pp. 10-22, 1976.

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

[H] H. B. Hunt III, On the time and tape complexity of languages, Ph.D. dissertation, Dept. of Computer Science, Cornell University, Ithaca, NY, 1973.

[HRS] H. B. Hunt III, S. S. Ravi, and R. E. Stearns, Separators, graphical homomorphisms, chromatic polynomials (extended Abstract), *Proc. 26th Ann. Allerton Conf. on Communications, Control, and Computing*, pp. 788-797, 1988.

[HS1] H. B. Hunt III and R. E. Stearns, Nonlinear algebra and optimization on rings are "hard," *SICOMP*, vol. 16, pp. 910-929, 1987.

[HS2] H. B. Hunt III and R. E. Stearns, The complexity of very simple Boolean formulas with applications, *SICOMP*, vol. 19, pp. 44-70, 1990. Also appears as Technical Report TR 87-23, Dept. of Computer Science, SUNY at Albany, Albany, NY, 1987.

[JP] D. S. Johnson and C. H. Papadimitriou, Computational complexity, in *The Traveling Salesman Problem*, ed. E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, pp. 37-86, Wiley, New York, 1985.

[K] R. M. Karp, reducibility among combinatorial problems, in *Complexity and Computer Computations*, ed. R. E. Miller and J. W. Thatcher, pp. 85-103. Plenum, New York, 1972.

[LT] R. L. Lipton and R. E. Tarjan, Applications of a planar separator theorem, *SICOMP*, vol. 9, pp. 615-629, 1980.

[MS] B. Monien and I. H. Sudborough, *Bounding the Bandwidth of NP-Complete Problems*, Lecture Notes in Computer Science, vol. 100, pp. 279-292, Springer Verlag, Berlin, 1981.

[P] C. H. Papadimitriou, The Euclidean traveling salesman problem is NP-complete, *Theoret. Comput. Sci.*, vol. 4, pp. 237-244, 1977.

[RH] S. S. Ravi and H.B. Hunt III, Application of planar separator theorem to counting problems, *Inform. Process. Lett.*, vol. 25, pp. 317-321, 1987.

[RS] N. Robertson and P. D. Seymour, Graph minors II, algorithmic aspects of tree-width, *J. Algorithms*, vol. 7, pp. 309-322, 1986.

[R1] J. M. Robson, A new proof of the NP completeness of satisfiability, *Proc. 2nd Australian Computer Science Conf.* pp. 62-69, 1979.

[R2] J. M. Robson, Subexponential algorithms for some NP-complete problems, unpublished manuscript, 1985.

[R3] A. Rosenthal, Dynamic programming is optimal for nonserial optimization problems, *SICOMP*, vol. 11, pp. 47–59, 1982.

[S1] S. K. Sahni, Some subexponentially recognizable *NP*-complete languages, Technical Report 74-14, Computer Science Dept., University of Minnesota, 1974.

[S2] T. J. Schaefer, The complexity of satisfiability problems, *Proc. 10th Ann. ACM Symp. on Theory of Computing*, pp. 216–226, 1978.

[S3] C. P. Schnorr, Satisfiability is quasi-linear complete in NQL, *J. Assoc. Comput. Mach.*, vol. 25, pp. 136–145, 1978.

[SH1] R. E. Stearns and H. B. Hunt III, On the complexity of the satisfiability problem and the structure of *NP*, Technical Report 86-21, Dept. of Computer Science, SUNY at Albany, Albany, NY, 1986.

[SH2] R. E. Stearns and H. B. Hunt III, Power indices and easier *NP*-complete problems, Technical Report 88-27, Dept. of Computer Science, SUNY at Albany, Albany, NY, 1988.

[SH3] R. E. Stearns and H. B. Hunt III, Structure trees and their application, Technical Report 90-2, Dept. of Computer Science, SUNY at Albany, Albany, NY, 1990.

[SM] L. J. Stockmeyer and A. R. Meyer, Word problems requiring exponential time, *Proc. 5th Ann. ACM Symp. on Theory of Computing*, pp. 1–9, 1973.

Received September 20, 1989, and in revised form January 31, 1990.