On Limited Nondeterminism and the Complexity of the V-C Dimension

Christos H. Papadimitriou*
University of California at San Diego, La Jolla, California 92093

and

Mihalis Yannakakis
AT&T Bell Laboratories, Murray Hill, New Jersey 07974

We characterize precisely the complexity of several natural computational problems in NP, which have been proposed but not categorized satisfactorily in the literature: Computing the Vapnik–Chervonenkis dimension of a 0–1 matrix; finding the minimum dominating set of a tournament; satisfying a Boolean expression by perturbing the default truth assignment; and several others. These problems can be solved in \( \rho^{O(\log^3 n)} \) time, and thus, they are probably not NP-complete. We define two new complexity classes between P and NP, very much in the spirit of MAXNP and MAXSNP. We show that computing the Vapnik–Chervonenkis dimension is complete for the weaker class, while the other two problems are complete for the more general class, while the other two problems are complete for the weaker class.

1. INTRODUCTION

Let \( \mathcal{C} \) be a family of subsets of some universe \( U \). The Vapnik–Chervonenkis dimension of \( \mathcal{C} \), \( d(\mathcal{C}) \) [VC], is the largest cardinality of a subset \( S \subseteq U \) such that the following holds: For all subsets \( T \subseteq S \) there is a set \( C(T) \in \mathcal{C} \) such that \( S \cap C(T) = T \). That is, all subsets of \( S \) are required to be present in \( \mathcal{C} \). The Vapnik–Chervonenkis dimension of \( \mathcal{C} \) is intuitively a measure of the “variability” of \( \mathcal{C} \). In fact, it has been proved that it is a reasonably precise estimate of the complexity of learning \( \mathcal{C} \), if \( \mathcal{C} \) is thought of as a class of concepts to be learned [BEHW]. But how hard is it to compute \( d(\mathcal{C}) \), given a family \( \mathcal{C} \) over a finite universe \( U \)?

This question was first addressed in [LMR] (the first author that paper suggested this problem to us). Although the problem is obviously in NP, there is no known polynomial-time algorithm for it; [LMR] note this open problem and present an interesting characterization of set families with \( d(\mathcal{C}) = 1 \).

Although it certainly starts like one, this is not an NP-completeness paper. Closer inspection reveals that the problem of computing the Vapnik–Chervonenkis dimension is very unlikely to be NP-complete. The reason is that, in the definition of the Vapnik–Chervonenkis dimension, the sought set \( S \) must satisfy \( 2^{|S|} \leq |\mathcal{C}| \), and hence \(|S| \leq \log |\mathcal{C}|\) (throughout this paper by \( \log n \) we mean \( \lceil \log_2 n \rceil \)). Therefore, the Vapnik–Chervonenkis dimension can be found in time \( O(n^{\log n}) \) in the length of the input. Still, it is important and intriguing to determine whether the problem can be solved in polynomial time. In this paper we give a precise characterization of the complexity of the problem V-C DIMENSION (“given \( \mathcal{C} \) and \( k \), is \( d(\mathcal{C}) \geq k \?”).

One can define many problems solvable within the same bound by simply restricting essentially any NP-complete optimization problem by a logarithmic bound in the objective function (see Section 2 for several examples). Not all of these problems are equivalent. It turns out that some can be solved in polynomial time, while others appear to be harder. But such problems, with the explicit mention of the logarithmic bound in their statement, are arguably artificial. However, there are other situations, besides V-C DIMENSION, in which the logarithm is not mentioned explicitly but appears for combinatorial reasons. Consider, for example, the following problem.

RICH HYPERGRAPH COVER. “Given a hypergraph \( H = (V, E) \) in which every hyperedge contains at least half of the nodes, find a minimum node cover.”

Let \( m \) be the number of hyperedges. It is not hard to see that hypergraph has a node cover of size \( \log m \). Take a node that belongs to at least half of the hyperedges (such a node is of course guaranteed to exist, just count the total appearances of nodes in hyperedges), put it in the cover, delete the hyperedges that contain the node, and repeat. This argument has appeared often in probabilistic contexts, for example, in Adleman’s proof that the class \( \text{R} \) has polynomial size circuits [A]. One can think of the nodes of the hypergraph as representing the sample points of a finite probability space and the hyperedges as representing events in that space. If each of \( m \) events has probability \( \frac{1}{3} \), then there are \( m \) sample points that cover all events. (And a somewhat larger set of sample points will cover each event not only once but a logarithmic number of times [CF].)
Meggido and Vishkin studied another very interesting problem of this nature, namely finding the smallest dominating set of a tournament [MV]. A dominating set in a directed graph is a set \( D \) of nodes such that for any node \( v \in D \) there is an edge \((u,v)\) with \( u \in D \); a tournament is a directed graph in which for any two nodes exactly one of two directed edges is present. It is easy to see that a tournament has a dominating set of size \( \log n \). Take a node that dominates at least half of the nodes (any tournament has one), put it in the set, delete the nodes it dominates, and repeat. Thus the problem TOURNAMENT DOMINATING SET, although it contains no explicit mention of \( \log n \), it still falls in the same pattern of \( \log n \) nondeterminism and exhaustive solution in \( O(n^{\log n}) \) time. Meggido and Vishkin prove [MV] that the problem \( \text{LOG}^2 \text{SAT} \) (satisfiability with \( \log^2 n \) variables) polynomially reduces to TOURNAMENT DOMINATING SET, and the latter in turn reduces to a generalization of \( \text{LOG}^3 \text{SAT} \) in which the formula has in its clauses, instead of literals, conjunctions of literals. They leave as an open problem in their paper whether there is a natural complexity class for which TOURNAMENT DOMINATING SET is complete. In this paper we introduce such a complexity class.

There is another problem in the literature, coming from artificial intelligence, which has a similar flavor [Se]. Suppose that we are given a Boolean expression \( \phi \), and by some default mechanism we have obtained a prototypical truth assignment \( T_0 \), which still fails to satisfy it. We must adjust the prototypical model to the situation at hand, as represented by \( \phi \). Since “most things should be normal,” it is natural to expect that there is a truth assignment that satisfies \( \phi \) and is close, in Hamming distance, to \( T_0 \). Selman asks whether there is a polynomial algorithm for finding a truth assignment within Hamming distance \( \log n \) of \( T_0 \), where \( n \) is the number of variables in \( \phi \). Let us call this problem \( \text{LOG ADJUSTMENT} \). Selman shows [Se] that LOGCLIQUE (is there a clique of size \( \log n \) in a given graph with \( n \) nodes?) is reducible to \( \text{LOG ADJUSTMENT} \), but provides no further complexity characterization of the problem.

There is a reason why the search for the complexity class appropriate for all these problems has been harder than usual: The computational phenomenon that must be captured is especially intriguing and novel. In each of the above problems, we have to choose \( \log n \) elements from a set of size \( n \). Since each element can be represented by \( \log n \) bits, that means we have to choose \( \log^2 n \) bits in all. However, what we have in these problems is not exactly \( \log^2 n \)-bounded nondeterminism, since the computation following the \( \log^2 n \) choices seems to be very restricted. It certainly appears much weaker than \( \text{DSPACE}(\log^2 n) \), as only the “first scan” over the work tape is nondeterministic (we discuss limited nondeterminism and \( \log^2 \) space in more detail in Section 5). A new dimension of complexity seems to be needed.

As in the similar case of the approximability of optimization problems [PY], we must turn to Fagin’s theorem [Fa] for inspiration. That result characterizes NP as precisely the class of all problems that can be put in this form

\[ I: \exists S \subseteq [n]^m \forall x \in [n]^p \exists y \in [n]^q \phi(I, S, x, y), \tag{1} \]

where \( I \subseteq [n]^m \) is the input relation, \( x \) and \( y \) are tuples of first-order variables ranging over \( [n] = \{1, 2, \ldots, n\} \), and \( \phi \) is a quantifier-free first-order expression involving the relation symbols \( I \) and \( S \), and the variables in \( x \) and \( y \). For example, SAT, can be written as \( \text{SAT} = \{ (P, N): \exists T \subseteq [2^n] \forall C \exists \exists \phi((P, N), T, C, v) \} \), where \( P \) and \( N \) (the positive occurrence relation and the negative occurrence relation) are two relations representing an expression in conjunctive normal form, and \( \phi \) is the first-order expression \( \exists (P, C, v) \land v \in T) \lor (N(C, v) \land v \notin T) \). That is, \( \phi \) states that either \( v \in T \) and \( v \) appears positively in clause \( C \), or \( v \notin T \) and \( v \) appears negatively in clause \( C \).

The classes we define in Section 2 are very much in the same spirit, except that the second-order structure \( S \) whose existence is asserted in \((1)\) is now of logarithmic size. We define two distinct classes, of which LOGNP shares the full quantificational pattern of \((1)\), while LOGSNP has one less alternation—notice the parallel with MAXNP and MAXSNP [PY]. In fact, we define our classes so that they are closed under reduction—which avoids misunderstandings that followed our less careful definition of MAXSNP. Furthermore, the classes become more stable, in the following sense: Different versions of a problem, with different encodings of the input as a relation \( I \), all become equivalent as long as they are polynomial-time intertranslatable. Alternatively, the first-order expression in the definition can be conveniently replaced by an arbitrary polynomial-time algorithm.

As was the case with the corresponding classes of optimization problems MAXNP and MAXSNP, the class LOGSNP is more dense in natural problems: It is not hard to see that LOGSNP contains the problems LOG CLIQUE, TOURNAMENT DOMINATING SET, RICH HYPERGRAPH COVER, LOG ADJUSTMENT, LOG CHORDLESS PATH, and several others. \( \text{We show that of these problems TOURNAMENT DOMINATING SET, RICH HYPERGRAPH COVER, and LOG ADJUSTMENT are LOGNP-complete} \) (Theorems 1, 2, and 3). The reduction to TOURNAMENT DOMINATING SET is a generic one, inspired by the reduction from \( \text{LOG}^2 \text{SAT} \) used in [MV]. We also show several reductions between logarithmic-cost versions of several optimization problems, and point out that the LOG version of NODE COVER is in \( P \).

But what about V-C DIMENSION, the problem that motivated us to look in this direction? In order to state V-C DIMENSION in this framework, the extra alternation of quantifiers in LOGNP is actually needed. That is, we must
state that there is a set $S$ of $k$ or more elements (where $k \leq \log n$) such that for all subsets $S'$ of $S$ there is a set $C$ in the family (this is still a "large" quantifier) such that for all indices $j$ (this is a universal "small" quantifier) $C$ and $S'$ agree. We show that V-C DIMENSION is LOGNP-complete (Theorem 5), thus settling the question of its precise complexity.

There is another class of problems solvable in time $n^{\log n}$, namely the problems associated with isomorphism of and generator set minimization in groups and their generalizations [Mi]. We argue in Section 5 that such problems are probably not in LOGNP; in fact, we show that the problem of telling whether a binary function has a set of $\log n$ generators is complete for the class NP[$\log^2 n$] of languages decided by Turing machines for which only the first $\log^2 n$ steps can be nondeterministic.

Comments on Motivation and Significance

We are not claiming that the calculation of $d(\emptyset)$ is a problem that researchers in learning theory are anxious to see solved efficiently. Complexity often has more subtle and interesting insights to offer, besides just which functions are hard to compute. For the same reason that efficient algorithms are direct results of nice mathematical structure (e.g., in linear algebra, matroids, and polyhedral combinatorics), classification in the highest possible complexity class is evidence that a concept, approach, or formulation is mathematically nasty, and infertile. In other words, complexity, besides its literal use, is also very valuable as a "metaphor". It is in this sense that understanding the complexity of V-C DIMENSION seems to us of importance. One way of interpreting our Theorem 5 is this: Although problems in physics have a natural measure of complexity (their (algebraic) dimension) which is rather immediately available, the corresponding natural concept for problems in cognition and learning appears to be substantially less accessible.

Fully understanding the complexity of a problem is proving it complete for a natural complexity class (but, of course, whether a class is natural to a large extent depends on the complete problems it contains and how natural they are). The identification of computational paradigms (complexity classes) with applications (computational problems) has been perhaps the most important methodological contribution of complexity theory to date—NP-completeness is, of course, the best-known example. Such results keep complexity theory grounded to relevance, while providing maximum information about the possibility of efficient solution (or, more generally, the "well-behavedness") of the problem under scrutiny.

Occasionally—as in the present paper, but also in past research [BM, Pa, PY, JPY]—the rare family of problems defying classification will require that a new computational paradigm, a new complexity class, be built around them. Defining such classes is not always unnecessary pollution of the landscape. Since these complexity classes contain, by the very way they were conceived, natural, important, well-studied complete problems, in some sense they existed before their definition, they are discovered, not invented. They represent computational paradigms that must be collapsed with known ones, if progress is to be made on understanding the complexity of the problems in hand. This is not theoretical possibility, as such proofs of collapse do come up [Sh, Pa, ALMSS], arguably motivated by the class definition.

### 2. LOGARITHMIC RESTRICTIONS OF NP

Recall the characterization of NP as all problems expressible as in (1). In this section we define two subclasses of NP by restricting the range of the structure $S$ in (1). In particular, we define LOGNP$_0$, (not yet our final goal) to be the class of all problems described as follows:

\[
\{ I : \exists S \in [n]^{\log n} \forall x \in [n]^{\log n} \phi(I, x, y, j) \}
\]

(compare with (1)). $S$ is now ordered subset $S = (s_1, \ldots, s_{\log n})$ of $[n]$. Notice the crucial detail that the single universally quantified variable $j$ ranges over $[\log n]$. $\phi$ is a quantifier-free first-order expression involving the relation symbol $I$ and the variables in $x, y$, as well as the variable $s_j$, where $j$ is the "small" variable. As with optimization problems, we can define an apparently weaker class LOGSNP$_0$. This class contains all problems definable by one less alternation of quantifiers.

\[
\{ I : \exists S \in [n]^{\log n} \forall x \in [n] \exists y \in [\log n] \phi(I, s_j, x, j) \}
\]

(compare with (1) and (2)). The "S" in LOGSNP$_0$ stands for "strict," reflecting the fact that these predicates make no use of the alternation allowed in NP.

We define LOGNP to be the class of all languages that can be polynomially reduced to a problem in LOGNP$_0$, and similarly for LOGSNP and LOGSNP$_0$. This closes these classes under reduction (and avoids misunderstandings such as those that followed our less careful definition of MAXSNP).

Observe that LOGSNP is contained in LOGNP. If (3) defines a problem $P$, then we can write the definition also in the form (2):

\[
\{ I : \exists S \in [n]^{\log n} \forall x \in [n] \exists y \in [\log n] \phi(I, s_j, x, j) \}
\]


Notice that we allow “small” variables (ranging over \([\log n]\)) among the “large” ones; this does not affect the power of our classes.

Closing the classes under reduction makes them more stable, in the following sense.

**Proposition 1.** The class of all problems defined as in (2), only with a polynomial-time algorithm replacing the first-order predicate \(\phi\) coincides with LOGNP. The same for (3) and LOGSNP.

**Proof.** We give the proof for LOGNP. The proof for LOGSNP is the same. Suppose first that a problem (that is, a set of structures \(I\)) is in the new class. We shall show that it belongs in LOGNP. From each \(I\) we can compute a new structure \(I'\) (this is the reduction part) which is the cartesian product of \(I\) with the set of all tuples \((s_j, x, y, j)\) such that \(\phi(I, s_j, x, y, j)\). It is trivial that the new problem is in LOGNP, since the polynomial-time \(\phi(I, s_j, x, y, j)\) part can now be replaced by an atomic formula. Hence the original problem was in LOGNP.

Conversely, suppose that a problem \(A\) is in LOGNP, i.e., there is a polynomial-time reduction \(f\) from \(A\) to a problem \(B\) that is described by form (2). Let \(\phi_B\) be the first-order expression for problem \(B\) in form (2). Then problem \(A\) can be defined also by (2) with the predicate \(\phi_B(f(I), s_j, x, y, j)\), which is clearly computable in polynomial time.

Proposition 1 has the additional desirable effect that we do not have to worry about the details of the input encoding; variants of the same problem with different input encodings are now in the class as well (see the proof of Theorem 1 below for examples).

The classes are robust in the sense that different variations in the definitions yield the same classes. We point out some of these variants now.

In (2) and (3) the set \(S\) is required to be of size exactly \(\log n\). It is easy to see however that problems whose input includes a parameter \(k\) (for example, is there a dominating set \(S\) of size \(k\)) with \(k<\log n\) can be expressed in this framework. For example, one way to do this is to exploit the fact that \(\phi\) is allowed to depend on the small index \(j\). We can set appropriately the truth value of \(\phi\) for values of the index \(j\) that are greater than \(k\), so that the elements of \(S\) beyond \(s_j\) do not matter; i.e., we let \(\phi\) be true in (2) and false in (3) if \(j>k\). (The parameter \(k\) can be included in the input structure \(I\) in various ways, for example, as the cardinality of a particular relation; the details are not important since \(\phi\) can be any polynomial time algorithm.)

We can pad the input if we wish, up to a polynomial; hence, replace \(n\) by a power \(n^c\), and thus have \(S\) range over sets of size \(c\log n\). Furthermore, we can interpret elements from \([n^c]\) as \(c\)-tuples with elements drawn from \([n]\). Thus, we can allow the solution \(S\) to be a set of pairs, triples, etc., instead of only a set of elements.

The solution \(S\) in definitions (2) and (3) is taken to be an ordered subset of \([n]\), i.e., the elements \(s_j\) are distinct. The classes do not change if we allow repetitions, i.e., if we let \(S\) be an ordered multiset (a list). This can be easily seen as follows. First, suppose that a problem can be expressed in form (2) or (3) using the list interpretation. We can express it with the set interpretation by taking \(\log n\) copies of the domain elements and letting the solution \(S\) use \(i\)th copy of \(s_j\), in \(i\)th position. Conversely, suppose that a problem can be expressed in form (3) (similarly with (2)) under the set interpretation. Add the following conjunct to (3): \(\forall i \in [n] \forall j' \in [\log n] \exists j \in [\log n] (j = i \lor j = j') \land (s_j \neq x)\). This forces all the elements of \(S\) to be distinct.

Let us now define the following problems:

- **V-C DIMENSION.** Given a finite family \(\mathcal{C}\) of finite sets, and an integer \(k\), is \(d(\mathcal{C}) \geq k\)?

- **LOG ADJUSTMENT.** Given a Boolean expression in conjunctive normal form with \(n\) variables, and a truth assignment \(T\), is there a satisfying truth assignment whose Hamming distance from \(T\) is at most \(\log n\)?

- **RICH HYPERGRAPH COVER.** Given a hypergraph \(H\) with \(n\) nodes, all edges of size at least \(n/2\), and an integer \(k\), does \(H\) have a node cover of size \(k\)?

- **LOG CLIQUE.** Given a graph with \(n\) nodes, does it have a clique of size \(\log n\)?

- **LOG NODE COVER.** Given a graph with \(n\) nodes, does it have a node cover of size \(\log n\)?

- **LOG DOMINATING SET.** Given a directed graph with \(n\) nodes, does it have a dominating set of size \(\log n\)?

- **TOURNAMENT DOMINATING SET.** Given a tournament with \(n\) nodes and integer \(k\), does it have a dominating set of size \(k\)?

- **LOG PATH.** Given a graph with \(n\) nodes, does it have a simple path of length \(\log n\)?

- **LOG CHORDLESS PATH.** Given a graph with \(n\) nodes, does it have a chordless path of length \(\log n\)?

- **LOG\(^2\) SAT.** Given a Boolean formula in conjunctive normal form with \(n\) clauses and \(\log^2 n\) variables, does it have satisfying truth assignment?

**Theorem 1.** (a) **LOG DOMINATING, TOURNAMENT DOMINATING SET, LOG NODE COVER, RICH HYPERGRAPH COVER, LOG ADJUSTMENT, LOG CLIQUE, LOG\(^2\) SAT, LOG PATH, and LOG CHORDLESS PATH are in LOGSNP.**

(b) **V-C DIMENSION is in LOGNP.**
Proof. (a) For some of the problems it is immediate that they can be written in the form required for this class. For example, LOG DOMINATING SET can be written as

\[ \exists S \in [n]^{\log n} \forall x \in [n] \exists j \in [\log n] [s_j = x \lor I(s_j, x)]. \]

Here \( I \) is the edge relation of the input graph. LOG NODE COVER, on the other hand, is

\[ \exists S \in [n]^{\log n} \forall x, y \in [n] \exists j \in [\log n] [x = s_j \lor y = s_j \lor \neg I(x, y)]. \]

TOURNAMENT DOMINATING SET. Strictly speaking, is this problem: Given a tournament and an integer \( k \), does the tournament have a dominating set of size at most \( k \)? By the “halving” argument in the introduction, we can restrict the problem to \( k \leq \log n \). We can incorporate the parameter \( k \) as explained in the comments after Proposition 1, by adding a conjunct \( j \leq k \) to the formula \( \phi \) above of LOG DOMINATING SET.

For RICH HYPERGRAPH COVER the input relation is the hyperedge containment: \( I(x, y) \) if and only if \( x \) is a node, \( y \) is a hyperedge, and \( x \in y \). The problem whether a hypergraph has a node cover of size \( \log n \) can be written as

\[ \exists S \subseteq [n]^{\log n} \forall y \in [n] \exists j \in [\log n] [I(s_j, y)]. \]

A trick similar to the one in the previous paragraph modifies this expression to encompass RICH HYPERGRAPH COVER.

LOG CLIQUE is a little more complicated:

\[ \exists S \subseteq [n]^{\log n} \forall x, y \in [n] \forall i, i' \in [\log n] \exists j \in [\log n] [I(x, y) \lor (i = j \land s_j \neq x) \lor (i' = j \land s_j \neq y)]. \]

Recall that “small” variables \( i, i' \) are allowed among the “large” ones \( x, y \).

To express LOG^2 SAT we need Proposition 1. Intuitively, the polynomial predicate \( \phi(I, s_j, x, j) \) now states “if the bits of \( s_j \) are interpreted as the truth values of variables \( (j - 1) \log n + 1 \) through \( j \log n \), then one of these variables satisfies clause \( x \).”

To express LOG PATH and LOG CHORDLESS PATH in this style would be interesting exercises, but at this point let us simply note that they both reduce to LOG^3 SAT. For LOG PATH, for example, suppose that the graph is \( ([n], E) \). Then the corresponding Boolean formula has variables \( x_{ij}, i, j = 1, ..., k = \log n \) for the bits of the nodes \( s_j, i = 1, ..., k = \log n \) on the path. The condition “nodes \( s_i \) and \( s_j \) should be adjacent” can be expressed in terms of variables \( x_{ij} \), \( i = 1, 2, j = 1, ..., k \) (in a manner depending on \( G \), of course). The most natural way to do this is in disjunctive normal form, with \(|E|\) disjuncts. Then we convert this condition to conjunctive normal form exhaustively, but in polynomial time (we have \( 2 \log n \) variables). We repeat for all conditions “nodes \( s_i \) and \( s_j \) should be adjacent, \( i = 1, ..., k = \log n - 1 \)" and for all conditions “nodes \( s_i \) and \( s_j \), \( i \neq j \) should be distinct” and similarly for LOG CHORDLESS PATH.

We can express LOG ADJUSTMENT as follows. Recall that by Proposition 1, \( \phi \) in form (2) is allowed to be an arbitrary polynomial-time computable predicate. Let \( B \) be the given CNF Boolean expression and \( T \) the given default truth assignment. To simplify the discussion, we can assume without loss of generality that \( T \) assigns the value false to all variables; just rename some variables by their negations if necessary. We represent a solution truth assignment that differs from \( T \) in positions \( v_1, ..., v_{\log n} \) in increasing order by a sequence \( S \) of \( \log n + 1 \) pair \( s_j = (z_j, w_j), j = 0, ..., \log n \), where \( z_j = y \) and \( w_j = y_{j+1} \) for all \( j \), except for the boundary cases \( z_0 = 0 \) and \( w_{\log n} = n + 1 \). (If the truth assignment differs in fewer positions then we can represent it either with a correspondingly shorter sequence \( S \), or pad the sequence to the same length with new dummy elements.) For \( S \) to be an acceptable solution it must meet the following requirements.

1. For every clause \( C \) of the formula \( B \) there is a \( j \) such that \( C \) contains positively \( z_j \), or contains negatively a variable between \( z_j \) and \( w_j \).
2. For every index \( i = 0, ..., \log n \), we have \( z_i < w_i \).
3. For every index \( i = 0, ..., \log n - 1 \), we have \( w_i = z_{i+1} \) (and \( z_0 = 0, w_{\log n} = n + 1 \)).

Requirements 1 and 2 are in the right format; note that each constraint involves only one tuple \( s_j \) of \( S \). Although requirement 3 concerns two elements of \( S \) we can express it also in the right format. For example, \( w_j \geq z_{j+1} \) can be written as: for all \( x \in [n] \), there is an index \( j \) such that \( (j = i \land x < w_i) \lor (j = i + 1 \land x \geq z_i) \). The other inequality \( w_i \leq z_{i+1} \) can be written similarly.

(b) For the V-C DIMENSION problem, the given input \( I \) consists of a family \( C \) of sets and an integer \( k \) and we wish to determine if \( \alpha(C) \geq k \). As we mentioned in the Introduction, if we have at most \( n \) sets drawn from a universe of size \( n \), then we may assume that \( k \leq \log n \). We can express V-C DIMENSION as

\[ \{ I : \exists S \subseteq [n]^{\log n} \forall x \in [n] \exists y \in [n] : \exists i \in [\log n] [\phi(I, s_j, x, y, j)] \}, \]

where the variable \( x \) is interpreted as ranging over all binary vectors of length \( k \) (ignore the last \( \log n - k \) bits of \( x \)), the variable \( y \) ranges over all sets of \( C \) and \( \phi(I, s_j, x, y, j) \) is the polynomial-time computable predicate “if \( j \leq k \) then \( s_j \in y \) if and only if the \( j \)th bit of \( x \) is one.”
3. LOGSNP-COMPLETE PROBLEMS

Theorem 2. LOG DOMINATING SET and TOUR-NAMENT DOMINATING SET are LOGSNP-complete.

Proof. The reduction showing completeness of LOG DOMINATING SET is a generic one. Let $A$ be a problem in LOGSNP of the standard form (3), and let $I$ be an input with underlying set $[n]$. We must create a graph $G = ([N], E)$ such that $I$ is a positive input of $A$ if and only if $G$ has a dominating set of size $\log N$. $G$ contains $n \log n$ nodes $u_{ij}$, $i = 1, \ldots, n$; $j = 1, \ldots, \log n$. We add all directed edges of the form $(u_{ij}, u_{i+1})$; that is, these nodes form log $n$ completely connected subgraphs with $n$ nodes each. We also have a node $v_x$ for each $x \in [n]^\alpha$. $G$ also has $m$ isolated nodes, so that $\log N = \log (n^\alpha + n \log n + m) = \log n + m$. Finally, there is an edge from $u_{ij}$ to $v_x$ if and only if for this particular value of $I$, for the tuple $x$, and for $i = s_j$, the expression $\phi(I, s_j, x, y)$ is satisfied.

Note that a dominating set of $G$ must contain all isolated nodes and at least one node $u_{ij}$ for each $j$, because the rest are not dominated by any other nodes. Suppose that $G$ has a dominating set of size log $N$. Since log $N = \log n + m$, this set must contain all isolated nodes, and exactly one $u_{ij}$ node for each $j$. Hence, such a dominating set exists if and only if there is a subset $S$ of the $u_{ij}$'s of size log $n$ that together cover all of the $v_x$'s, which is equivalent to saying that $I$ is a positive instance of problem $A$.

Modifying the above construction, along the lines of a construction presented in [MV], establishes that TOUR-NAMENT DOMINATING SET is LOGSNP-complete. In particular, the construction must be modified so that the directed graph produced is a tournament. Let $T$ be a tournament with minimum dominating set of size log $n + 2$ [MV] show how to construct such a tournament with $n$ nodes. In addition to the previous nodes, our graph now contains, for each $j = 0, \ldots, \log n$, a copy $T_j$ of the tournament, call is nodes $t_{ij}$, $i = 1, \ldots, p$. The edges within each copy are exactly as in $T$. Across different copies, $(t_{ij}, t_{i'})$ is an edge for $j \neq j'$ and $i \neq i'$, iff $(t_j, t_{i'})$ is an edge of $T$. But if $i = i'$, then $(t_{ij}, t_{i'})$ is an edge iff $j < j'$.

Thus, the copies of $T$ now form a tournament, but the rest of the graph does not. There are several pairs of nodes that are not connected by an edge; for each such pair we must decide which of the two beats the other—that is, becomes the tail of the edge. First, all $t_{ij}$ nodes beat all $v_x$ nodes. All $u_{ij}$ nodes that do not beat $v_x$ are beaten by it. Each node $t_{ij}$ beats all nodes $u_{ij'}$ with $j \neq j'$. And each node $u_{ij}$ beats all nodes $t_{ij'}$. Finally, the $u_{ij}$ nodes are connected in an arbitrary way (here we have to delete edges from the complete graph), and similarly for the $v_x$ nodes. Finally, we add a new node, call it $u_{00}$, which beats all nodes except for the $t_{ij}$’s, $j > 0$. The question is whether there is a dominating set of size $k = \log n + 1$.

We claim that the resulting tournament has a dominating set of size $k$ if and only if the original instance $I$ was a “yes” instance of the generic problem in LOGSNP. The reason is the following: Suppose that there is a $j$ (possibly 0) such that no $u_{ij}$ node participates in the dominating set. Then all nodes of the $j$th copy of $T$ must be dominated by nodes within the copies of $T$. The $j$ indices of these nodes must therefore comprise a dominating set of $T$, and thus there must be at least log $n + 2$ of them, which is absurd. So, the log $n + 1$ members of the dominating set are distributed one at each level of the $u_{ij}$'s. In particular, $u_{00}$ is in the dominating set, and thus all $u_{ij}$'s are dominated. Hence the argument in the proof for LOG DOMINATING SET establishes that there is a dominating set of size $k$ if and only if the original instance $I$ was a positive one.

Theorem 3. RICH HYPERGRAPH COVER is LOGSNP-complete.

Proof. Notice that LOG DOMINATING SET in a graph $G = ([V], E)$ is a node cover problem in a hypergraph $H$: For each node $v$ define the hyperedge $e_v = \{u : u = v\text{ or } (u, v) \in E\}$. Then the dominating sets of $G$ coincide with the node covers of $H$. However, the edges $e_v$ may now contain fewer than half the nodes. We shall modify the structure by taking copies of the edges and adding new nodes to enlarge them in such a way that each edge contains at least half the nodes but the new nodes do not help in forming a smaller node cover.

Let $l > 2 \log n + 2$, and $r = (2^l - 1)^2$. We interpret an element of $[r]$ as a binary vector of the form $a_1a_2$, where $a_1$ and $a_2$ are nonzero vectors of length $l$. We add new nodes $u_1, \ldots, u_n$, and replace every edge $e_v$ of $H$ by $r$ edges $e_{v_1}, \ldots, e_{v_r}$ which contain the same set of original nodes as $e_v$. The inclusion of new nodes to the edges is determined as follows: Let $u_0$ be a new node and let $e_{v_j}$ be an edge, where $j$ corresponds to the vector $a_1a_2$ and $j$ corresponds to the vector $b_1b_2$; then node $u_a$ belongs to the edge $e_{v_j}$ if the inner product $a_1 \cdot b_1 = 1$ or $a_2 \cdot b_2 = 1$ where the arithmetic is in GF (2). Let $H'$ be this hypergraph.

By the definition, every edge of $H'$ contains $\frac{1}{2}$ of the new nodes $u_i$; therefore it certainly contains more than half of all the nodes. Clearly, if $H$ contains a node cover of size log $n$, then also the hypergraph $H'$ contains the same node cover. Conversely, consider any node cover $C$ of $H'$ that has less than $l$ nodes. We will argue that the old nodes in $C$ form a node cover of $H$. Thus, $H'$ has a node cover of size log $n$ if and only if $H$ does. Suppose that some edge $e_v$ of $H$ is not covered by any (old) node of $C$. Consider the vectors $a_1a_2$ corresponding to the indices of the new nodes $u_i$ of $C$. Since $|C| < l$, there are less than $l$ values of $a_1$ and $a_2$. Hence, there are nonzero vectors $b_1$ and $b_2$ of length $l$ such that $a_1 \cdot b_1 = 0$ and $a_2 \cdot b_2 = 0$ for all members $u_i$ of $C$. But then consider the edge $e_v$ of $H'$, where $j$ corresponds to $b_1b_2$. By the construction, $e_v$ is not covered by any old or new element of $C$, a contradiction.
Corollary 1. LOG ADJUSTMENT is LOGSNP-complete.

Proof. Notice that LOG HYPERGRAPH COVER is a special case of LOG ADJUSTMENT in which all clauses have only positive literals, and the default truth assignment \( T \) gives the value false to all the variables. The result follows then from Theorem 3.

We do not know whether LOG\(^2\)SAT, LOG CLIQUE, or LOG CHORDLESS PATH are LOGSNP-complete. However, we can show:

Theorem 4. LOG CLIQUE and LOG CHORDLESS PATH are polynomially equivalent.

Proof. For the reduction from LOG CLIQUE to LOG CHORDLESS PATH, suppose that we are given a graph \( G = (V,E) \) with \( n \) nodes. We construct a graph \( G' \) as follows: First, \( G' \) has \( \log n \) disjoint copies of \( V \); the \( j \)th copy \( V_j \) has nodes \( c_{ij}, i = 1, ..., n \). Two nodes \( c_{ij} \) and \( c_{i'j'} \) are connected in \( G' \) if and only if \( i = i' \) or \( j = j' \) or \([i, i'] \notin E \). Finally, for all \( j < \log n \) we have a path of length two \([p_{j1}, p_{j2}, p_{j3}] \) and edges from all nodes of \( V_j \) to \( p_{j1} \) and from \( p_{j3} \) to all nodes of \( V_{j+1} \). We also add to \( G' \) enough isolated nodes to bring the total of nodes for \( G' \) to \( N = \frac{3}{2} n^4 \). We claim that there is a chordless path of length \( \log N = 4(\log n - 1) \) — that is, \( G' \) is a positive instance of LOG CHORDLESS PATH — iff \( G \) has a clique of size \( \log n \). In proof, if \( G \) has a clique of size \( \log n \), then by taking a copy of its nodes, one from each copy of \( V \), and connecting them in order via the paths of length four, we form a chordless path of length \( 4(\log n - 1) \) nodes. Conversely, suppose that \( G' \) has a chordless path \( P \) of length \( 4(\log n - 1) \). Since every copy \( V_j \) of \( V \) induces a clique, \( P \) cannot contain more than two nodes from the same copy, and if it contains two nodes then it cannot contain the nodes \( p_{j1}, p_{j2}, p_{j3} \) of the following (and the preceding) path. It follows easily from this observation that for \( P \) to have length \( 4(\log n - 1) \), it must contain all the length-two paths and exactly one node from each copy of \( V \). Then the \( i \) indices of the nodes of \( P \) in the copies of \( V \) must form a clique of the graph \( G \) and there are \( \log n \) of them.

For the other direction, given an instance \( G = (V,E) \) of LOG CHORDLESS PATH, we construct a graph \( G' \) which contains again \( \log n \) copies of \( V \). Every copy \( V_j \) induces an independent set. Two nodes \( c_{ij}, c_{i'j'} \) are connected by an edge iff \([i, i'] \notin E \) and \( j \neq j' \). Two nodes \( c_{ij}, c_{i'j'} \) from two nonconsecutive copies \( j \neq j' > 1 \) are connected by an edge iff \([i, i'] \notin E \) and \( i \neq i' \). Add \( m \) additional nodes and connect them among themselves to all other nodes, so that the total number \( N = n \log n + m \) of nodes satisfies \( \log N = \log n + m \). We claim that \( G \) has a chordless path with \( \log n \) nodes if and only if \( G' \) has a clique with \( \log N \) nodes. First, if \( G \) has such a path, then we let the clique contain for each \( j = 1, ..., \log n \), the \( j \)th copy of the \( j \)th node of the path, and the \( m \) additional nodes. Conversely, suppose that \( G' \) contains a clique \( C \) of \( \log N \) nodes. The clique can contain at most one node from every copy \( V_j \), because \( V_j \) induces an independent set. Since \( |C| = \log n + m \), \( C \) must contain exactly one node from each \( V_j \) plus the \( m \) additional nodes. By construction, the nodes of \( G \) corresponding to the \( \log n \) nodes of \( C \) from the copies of \( V \) form a chordless path.

We already know that both LOG CLIQUE and LOG CHORDLESS PATH polynomially reduce to LOG\(^2\)SAT (recall the proof of Theorem 1). As for LOG NODE COVER we have:

Theorem 5. LOG NODE COVER is in \( P \).

Proof. The nodes in any maximal matching constitute a node cover at most twice the optimum. To determine whether the minimum node cover of a graph \( G = (V,E) \) is at most \( \log n \) first find such an approximate node cover \( M \); if it is larger than \( 2 \log n \) quit. Otherwise, recall that \( V \setminus M \) is an independent set. Therefore, the optimum node cover \( C \) consists of two parts: \( C \cap M \) plus all nodes in \( V \setminus M \) that are adjacent to some node outside \( C \cap M \). Therefore \( C \) can be found by iterating over all possible \( C \cap M ' s \), that is, all possible subsets of \( M \). Since \( |M| \leq 2 \log n \), we know that there are at most \( n^2 \) such subsets, and the algorithms is \( O(n^3) \).

Intriguingly, the LOG PATH problem seems to be somewhere in between. A result due to Monien [Mo] suggests that we can determine whether a graph has a path of length \( \log n \) in time \( O((\log n)!)= O(n^{\frac{3}{2} n^4}) \), and so the problem seems to belong to some intermediate class LOGLOGNP.

4. The Complexity of the V-C Dimension

In this section we prove the following result.

Theorem 6. V-C Dimension is LOGNP-complete

Proof. By a generic reduction. Let \( A \) be a problem in LOGNP in the standard form (2). We shall reduce it to V-C Dimension. That is, given any input \( A \) we shall construct a family of sets \( \mathcal{G} \subseteq 2^U \) and integer \( k \) such that there is a set \( \mathcal{T} \subseteq U \) of cardinality \( k \) such that each subset of \( \mathcal{T} \) is represented in some set in \( \mathcal{G} \) if and only if \( P \) is a positive instance of \( A \). In fact, we shall represent \( \mathcal{G} \) by its set-element adjacency matrix \( C \), where columns correspond to elements and rows correspond to sets.

\( C \) has a total of \( (n+2+2p+q) \log n \) columns, where \( n \) is the number of elements in the instance \( I \) and \( p \) and \( q \) are the parameters of the definition of \( A \) as in (2). The columns are arranged in order of length 4, groups of \( \log n \) “blocks” of \( n \) columns each (the \( i \)th such
group will correspond to $s_j$, followed by an "identification section" with $p \log n$ columns, and finally a "tail section" with $(2+p+q) \log n$ additional columns. Consider a value $x$ of the variable vector $x \in [n]^p$ (these values are the "clauses" in our problem); $x$ can be thought of as an element of $\{0,1\}^{p \log n}$. For each such $x$ we have in our matrix a set $R_x$ of rows of $C$. For every $(z_1, \ldots, z_{\log n}) \in \{0,1\}^{\log n} - \{(1,1,\ldots,1)\}$, $R_x$ contains a row with all $n$ elements in the $i$th block of columns having equal value $z_i$. In addition, for each value $y \in [n]^q$ (these are the "literals") we have one row of $R_y$, which is 1 at the $i$th column of the $i$th block if and only if $\phi(i, s_j, x, y, j)$ is true whenever $s_j = i$. The identification sections of all rows in $R_y$ consist of $x$ (considered a $p \log n$-long bit vector that identifies the row as one of $R_y$) followed by the tail of $(2+p+q) \log n$ zeros. Finally, besides all the $R_y$'s, matrix $C$ contains some "additional" rows, one for each $(z_1, \ldots, z_{\log n}) \in \{0,1\}^{\log n}$ (where the $i$th block has all entries equal to $z_i$), for each identification $x$, and for each tail $y \not\equiv (2+p+q) \log n$.

It is not hard to see that, if $I$ is indeed a positive input of $A$, then the Vapnik–Chervonenkis dimension of $C$ is at least (in fact, exactly) $(3+2p+q) \log n$. Let $S = (s_1, \ldots, s_{\log n})$ be a solution to the instance $I$ of $A$ satisfying (2). The columns that constitute the set $T$ pick the $i$th column from each block, plus the last $(2+2p+q) \log n$ columns. All combinations of values for these columns are present: For each combination of values $(z_1, \ldots, z_{\log n}) \in \{0,1\}^{\log n}$ and each $x$ for the identification section and $i$ for the tail, if the $z$'s are not all ones or the $i$'s are not all zeros, then the appropriate row exists among the additional rows or the first rows of $R_y$. In the remaining case, we must determine for this value of $x$ a value of $y$ such that for all $i$ the $y$'s satisfy $\phi$; but this is possible by the fact that $S$ is a solution for the positive instance $I$.

The converse is a little more complicated. Suppose that the dimension of $C$ is at least $(3+2p+q) \log n$. The number of distinct rows if one ignores the tail is $n^{1+2p+q} (n$ values of $x$, $n^p$ values of $y$ for each of the $n^p$ $R_y$'s, plus $n^q$ values for the identification section). Thus, there can be at most $(1+2p+q) \log n$ columns outside the tail in the set $T$ of columns that realizes the dimension. Hence, $T$ must contain at least two columns from the tail section.

Suppose that $T$ also contains, besides these two columns from the tail section, two columns from the $i$th block. Then the combination 0111 cannot be achieved of these four columns. Hence, $T$ contains at most one column from each block. It follows that $T$ must contain exactly one column from each block plus all the $(2+2p+q) \log n$ columns from the identification and the tail section. Thus, the set $T$ induces a solution $S = (s_1, \ldots, s_{\log n})$ for the instance $I$ of $A$: $s_i$ is the index of the column of $T$ from the $i$th block, for each $i$. For every $x \in [n]^p$ there is a row of the matrix whose identification columns spell the vector $x$, the tail columns have value 0, and the columns from the first section (the blocks) have value 1. This row belongs to $R_x$, and corresponds to a $y \in [n]^q$ such that $\phi(i, s_j, x, y, j)$ is true for all $i$. Therefore, the solution $S$ satisfies (2), and $I$ is a positive input to $A$.

5. Further Questions

One can define subclasses of NP by restricting the number of nondeterministic steps; in particular, Kintala and Fischer and Díaz and Torán studied classes that use polylogarithmic nondeterminism [KF, DT]—an attempt to delimit nondeterminism even lower, at the level of NC, resulted in a deterministic space class; see [Wo]. Let $NP[\log^2 n]$ denote the subclass of NP in which only the $\log^2 n$ first steps are nondeterministic. (This class was denoted $\beta_2$ in [DT]). It is clear that

$LOGSNP \subseteq LOGNP \subseteq NP[\log^2 n] \cap DSPACE(\log^2 n)$.

However, LOGNP appears weaker than both $NP[\log^2 n]$ and DSPACE($\log^2 n$). First, DSPACE($\log^2 n$) is equivalent to unbound alternations of quantifiers of the form $\exists S_1 \in [n]^{|S_1|^q} \forall S_2 \in [n]^{|S_2|^q} \phi$ (compare with (2) and (3) of Section 1). As for $NP[\log^2 n]$, it allows for a general deterministic polynomial-time computation after the nondeterministic phase, instead of the evaluation of a first-order expression with limited quantifier alteration allowed in LOGNP.

Miller studied in [MI] an intriguing class of computational problems of an algebraic nature: Finding generators and testing isomorphism of groups and Latin squares. Let $f: [n]^2 \rightarrow [n]$ be a binary function. A set $S \subseteq [n]$ is called a set of generators if any element of the universe $[n]$ can be expressed as a tree of applications of $f$ to elements of $S$ (in the case of groups or quasigroups, we can use also their inverses). Consider the following problem.

LOG GENERATORS. Given the multiplication table of the binary function $f$ on $[n]$, does $f$ have a set of generators of cardinality $\log n$?

Now, if it so happens that the multiplication table of $f$ is a Latin square (that is, $f$ has unique left and right inverses), then $f$ always has a set of generators of cardinality $\log n$; that is, for simple algebraic reasons any nonredundant set of generators is guaranteed to have $\log n$ or fewer elements, and thus the corresponding problem need not mention $\log n$ (compare with the dominating set problem and tournaments).

MINIMUM GENERATOR SET OF QUASIGROUP. Given an $n \times n$ Latin square and an integer $g$, is there a set of $g$ elements that are generators?

A related problem is the following.
QUASIGROUP ISOMORPHISM. Given two \( n \times n \) Latin squares, are they isomorphic?

All three problems are in \( \text{NP}[\log^2 n] \). In fact, we can show the following.

**Theorem 7.** \( \text{LOG GENERATORS} \) is \( \text{NP}[\log^2 n] \)-complete.

**Proof.** We start from the following \( \text{NP}[\log^2 n] \)-complete problem: We are given a monotone circuit with \( n \) gates and \( \log^2 n \) inputs (and their negations), and we are asked whether there is a set of inputs such that the circuit accepts.

We assume without loss of generality that the circuit has fan-out 2. We partition the inputs into \( \log n \) blocks of \( \log n \) inputs each.

The domain \( D \) of the binary function \( f \) contains one element for each gate, one element for each input variable and for its complement, elements \( s_1, \ldots, s_{\log n}, t \), and the identity \( \text{id} \); and finally it contains sets \( Q_1, \ldots, Q_{\log n} \), each with \( n \) elements, corresponding to the blocks of input variables.

We regard each member \( u \) of \( Q_i \) as a binary vector of length \( \log n \) and associate it with a truth assignment for the input variables of the \( i \)-th block.

The multiplication table \( f \) is defined as follows. For every \( a \in D \), \( f(a, \text{id}) = f(\text{id}, a) = a \). All the other products are equal to \( \text{id} \), except for the following cases:

1. If \( u \in Q_i \) and the \( j \)-th bit of \( u \) (as a binary number) is 1 (resp. 0) then \( f(u, s_j) \) is equal to the \( j \)-th input variable of the \( i \)-th block (resp. its negation).

2. If \( a \) is an AND gate with inputs \( b \) and \( c \), then \( f(b, c) = a \).

3. If \( a \) is OR gate with inputs \( b \) and \( c \) and the edge \((b, a)\) of the circuit is the first (resp. second) edge out of \( b \) then \( a = f(b, i) \) (resp. \( a = f(i, b) \)), and similarly for the input \( c \).

4. Let \( o \) be the output gate of the circuit. Arrange the elements of each \( Q_i \) in a cycle, and for each \( u \in Q_i \), let \( f(u, o) \) be the successor of \( u \) on the cycle.

We claim that there is an input for which the circuit accepts iff \( 2 \log n + 1 \) generators suffice. For the one direction, assume that there is an accepting input assignment: Let \( S \) be the set consisting of \( s_1, \ldots, s_{\log n}, t \), and one element from each \( Q_i \) corresponding to the accepting input assignment. It is easy to see that \( S \) generates all input literals that have value 1 (by part 1 of the construction), and then show by a straightforward induction (by parts 2 and 3) that \( S \) generates all gates that have value 1, and thus, in particular, the output gate \( o \). By part (4), \( S \) can generate then all elements of all \( Q_i \), corresponding to all possible input assignments. After that, one can generate all the elements of \( D \) using 1, 2, and 3.

For the other direction, observe that no multiplication generates \( s_1, \ldots, s_{\log n}, t \). Furthermore, every generation of a member of \( Q_i \) involves another member of \( Q_i \). Therefore, every set \( S \) of generators must contain \( s_1, \ldots, s_{\log n}, t \) and at least one element from each \( Q_i \). If \( |S| = 2 \log n + 1 \) then it cannot have any more elements and must contain exactly one member of \( Q_i \) for each \( i \). These elements define an input assignment \( x \). If \( x \) is not an accepting input, then \( S \) can only generate the true input literals and the true gates, and thus it cannot generate, in particular, the output gate \( o \). Therefore, \( x \) must be an accepting input assignment.

We conjecture that this result also holds for the more structured MINIMUM GENERATOR SET OF A QUASIGROUP problem. In contrast, QUASIGROUP ISOMORPHISM was recently shown to be in \( \text{DSPACE}(\log^2 n) \) \[Wo\]. Notice that the corresponding problems for groups were known to be in \( \text{DSPACE}(\log^2 n) \) \[LSZ\].

There are of course many more classes that we could consider, with more and different alternations of quantifiers; but in the absence of natural problems in them there is little point or motivation to do so. As for the omissible class \( \text{LOGLOGNP} \) containing \( \text{LOG PATH} \), we conjecture that \( \text{LOG PATH} \) is indeed in \( \text{P} \).

One could restrict the number of elements in a solution \( S \) in (2) even further than \( \log n \), to a constant. We suspect that this may give an alternative formulation of the fixed-parameter problems considered in \[AEFM, DF\].

**Note added in proof.** The conjecture that \( \text{LOG PATH} \) is in \( \text{P} \) was proved by N. Alon, R. Yuster, and U. Zwick (1994 STOC).

**REFERENCES**


