Complexity of Problems on Graphs Represented as OBDDs*

J. Feigenbaum,[1] S. Kannan,[2] ** M. Y. Vardi,[3] *** M. Viswanathan[2] †

[1] AT&T Labs – Research
Room C203, 180 Park Avenue
Florham Park, NJ 07932 USA
jf@research.att.com
[2] Computer and Information Sciences
University of Pennsylvania
Philadelphia, PA 19104 USA
kannan@central.cis.upenn.edu
maheshv@gradient.cis.upenn.edu
[3] Computer Science
Rice University
Houston, TX 77251 USA
vardi@cs.rice.edu

Abstract. To analyze the complexity of decision problems on graphs, one normally assumes that the input size is polynomial in the number of vertices. Galperin and Wigderson [13] and, later, Papadimitriou and Yannakakis [18] investigated the complexity of these problems when the input graph is represented by a polylogarithmically succinct circuit. They showed that, under such a representation, certain trivial problems become intractable and that, in general, there is an exponential blow up in problem complexity.

In this paper, we show that, when the input graph is represented by a small ordered binary decision diagram (OBDD), there is an exponential blow up in the complexity of most graph problems. In particular, we show that the GAP and AGAP problems become complete for PSPACE and EXP, respectively, when the graphs are succinctly represented by OBDDs.

* An extended abstract of this paper appears in the Proceedings of the 1998 Symposium on Theoretical Aspects of Computer Science.

** Work done in part as a consultant to AT&T and supported in part by NSF grant CCR96-19910 and ONR Grant N00014-97-1-0505.

*** Work done as a visitor to DIMACS and Bell Laboratories as part of the DIMACS Special Year on Logic and Algorithms and supported in part by NSF grants CCR-9628400 and CCR-9700061 and by a grant from the Intel Corporation.

† Supported by grants NSF CCR-9415346, NSF CCR-9619910, AFOSR F49620-95-1-0508, ARO DAAH04-95-1-0092, and ONR Grant N00014-97-1-0505.

1 Introduction

The efficiency of algorithms is generally measured as a function of input size [11]. In analyses of graph-theoretic algorithms, graphs are usually assumed to be represented either by adjacency matrices or by adjacency lists. However, many problem domains, most notably computer-aided verification [7,9,16], involve extremely large graphs that have regular, repetitive structure. This regularity can yield very succinct encodings of the input graphs, and hence one expects a change in the time- or space-complexity of the graph problems.

The effect of succinct input representations on the complexity of graph problems was first formalized and studied by Galperin and Wigderson [13]. They discovered that, when adjacency matrices are represented by polylogarithmically-sized circuits, many computationally tractable problems become intractable. Papadimitriou and Yannakakis [18] later showed that such representations generally have the effect of exponentiating the complexity (time or space) of graph problems. Following this line of research, Balcázar, Lozano, and Torán [1–3,22] extended these results to problems whose inputs were structures other than graphs and provided a general technique to compute the complexity of problems with inputs represented by succinct circuits. They characterized the class of problems that become intractable when inputs are represented in this way. Veith [23,24] showed that, even when inputs are represented using Boolean formulae (instead of circuits), a problem's computational complexity can experience an exponential blow-up.

The possibility of representing extremely large graphs succinctly has attracted a lot of attention in the area of computer-aided verification [7,9,16]. In this domain, graphs are represented by *ordered binary decision diagrams* (OBDDs). OBDDs are special kinds of rooted, directed acyclic graphs that are used to represent Boolean circuits. Because of their favorable algorithmic properties, they are widely used in the areas of digital design, verification, and testing [8,9,17]. Experience has shown that OBDD-based algorithmic techniques scale up to industrial-sized designs [10], and tools based on such techniques are gaining acceptance in industry [4]. Although OBDDs provide canonical succinct representations in many practical situations, they are exponentially less powerful than Boolean circuits in the formal sense that there are Boolean functions that have polynomial-sized circuit representations but do not have subexponential-sized OBDD representations [19,20]. (On the other hand, the translation from OBDDs to Boolean circuits is linear [7].) Thus, the results of [2,3,13,18,22–24] do not apply to OBDD-represented graphs. Furthermore, even though Boolean formulae are, in terms of representation size, less powerful than circuits, they are still more succinct than OBDDs. Translation from OBDDs to formulae leads to at most a quasi-polynomial ($n^{\log n}$) blow-up, whereas there are functions (*e.g.*, multiplication of binary integers) that have polynomial-sized formulae but require exponential-sized OBDDs. Indeed, while the satisfiability problem is NP-complete for Boolean formulae, it is in nondeterministic logspace for OBDDs [7]. Therefore, the results in [23,24] do not apply to our case.

In this paper, we show that, despite these theoretical limitations on the power of OBDDs to encode inputs succinctly, using them to represent graphs nonetheless causes an exponential blow-up in problem complexity. That is, the well-studied phenomenon of exponential increase in computational complexity for graph problems with inputs represented by Boolean circuits or formulae [2,3,13,18,22–24] also occurs when the graphs are represented by OBDDs. Graph properties that are ordinarily NP-complete become NEXP-complete. The Graph Accessibility Problem (GAP) and the Alternating Graph Accessibility Problem (AGAP) for OBDD-encoded graphs are PSPACE-complete and EXP-complete, respectively. Both GAP and AGAP are important problems in *model checking*, a domain in which OBDDs are widely used [9,12,14,15].

In section 2, we formally define OBDDs and present some known results about them. In section 3, we discuss the problem in greater detail and compare Papadimitriou and Yannakakis's result to ours. Finally, in sections 4-6, we give our technical results.

2 Preliminaries

Definition 1. A *Binary Decision Diagram* (BDD) is a single-rooted, directed acyclic graph in which

- Both are external nodes labeled 0, or
- Both are external nodes labeled 1, or
- Each internal node (*i.e.*, a node with nonzero outdegree) is labeled by a Boolean variable.
- Each internal node has outdegree 2. One of the outgoing edges is labeled 1 (the "then-edge") and the other is labeled 0 (the "else-edge").
- Each external node (*i.e.*, a node with zero outdegree) is labeled 0 or 1.

Let $X = \{x_1, x_2, \ldots, x_n\}$ be the set of Boolean variables that occur as labels of nodes in a given BDD B. Each assignment $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_n)$ of Boolean values to these variables naturally defines a computational path – the one that leads from the root to an external node and has the property that, when it reaches a node labeled x_i, it follows the edge labeled α_i, for any i.

Definition 2. Two nodes u and v of a BDD are *equivalent* if

- u and v are the same node, or
- The label of u is the same as the label of v. Furthermore, u_1 is equivalent to v_1, where $< u, u_1 >$ and $< v, v_1 >$ are the then-edges of u and v, respectively, and u_0 is equivalent to v_0, where $< u, u_0 >$ and $< v, v_0 >$ are the else-edges of u and v, respectively.

A BDD in which no two nodes are equivalent is called *reduced*.

Definition 3. Let $<$ be a total ordering on a set X. An *Ordered Binary Decision Diagram* (OBDD) over $(X, <)$ is a reduced BDD with node-label set X such that,

along any path from the root to an external node, there is at most one occurrence of each variable, and the order in which the variables occur along the path is consistent with the order $(X, <)$. The *size* of an OBDD is the number of internal nodes in it.

Definition 4. An OBDD O *represents the Boolean function* $f(x_1, x_2, \ldots, x_n)$ if, for each assignment $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_n)$ to the variables of f, the computation path defined by α terminates in an external node that is labeled by the value $f(\alpha_1, \alpha_2, \ldots, \alpha_n)$.

An OBDD O *represents the graph* $G = (V, E)$ if O represents the Boolean function adj, where

$$adj(v_1, v_2) = \begin{cases} 1 \text{ if and only if } < v_1, v_2 > \in E \\ 0 \text{ otherwise} \end{cases}$$

Theorem 5 (Bryant [7]). *For each Boolean function f and ordering $(X, <)$ of the set of variables X, there is a unique (up to isomorphism) OBDD over $(X, <)$ that represents f.*

Theorem 6 (Bryant [7]). *Let F and G be OBDDs over $(X, <)$ representing functions f and g, respectively. Let the size of F be m, the size of G be n, and $< op >$ be any Boolean operation. Then there is an OBDD over $(X, <)$ of size at most mn and constructable in time polynomial in m and n that represents $f < op > g$.*

Definition 7. Let $L = (G, <)$ be a linear order on the gates of a circuit, where the inputs and outputs are classified as special instances of gates. We say that the *forward cross section of the circuit at gate g* is the set of wires connected to the output of some gate g_1 and an input of some gate g_2 such that $g_1 \leq g$ and $g < g_2$. The *reverse cross section of the circuit at gate g* is the set of wires connected to an output of some gate g_1 and an input of some gate g_2 such that $g_2 \leq g$ and $g < g_1$.

Definition 8. The *forward width of a circuit under order L*, denoted w_f, is the maximum, over all gates g, of the forward cross section at g. Similarly, *the reverse width of the circuit under order L*, denoted by w_r, is the maximum, over all gates g, of the reverse cross section at g.

Theorem 9 (Berman [5]). *For a circuit and gate-ordering with $w_r = 0$, there exists a variable ordering such that the OBDD size is bounded by $n2^{w_f}$, where n is the number of inputs to the circuit.*

Notation: We will be interested in complexity classes $\mathbf{C}$ that have universal Turing machines and complete problems. Let $U_{\mathbf{C}}$ denote the Universal Turing machine for the complexity class $\mathbf{C}$. Let $\mathcal{L}(U_{\mathbf{C}})$ be the language accepted by the machine $U_{\mathbf{C}}$ *i.e.*, $\mathcal{L}(U_{\mathbf{C}}) = \{x : x$ encodes a $\mathbf{C}$-bounded Turing machine M and an input y such that M accepts $y\}$.

For an n-bit number x, we will refer to the i^{th} bit by $x^{(i)}$, where $x^{(n)}$ is the most significant bit.

3 Problem Statement

Papadimitriou and Yannakakis [18] show that any NP-complete graph property π to which satisfiability is reducible by a *projection*, in the sense of Skyum and Valiant [21], becomes NEXP-complete when problem instances are encoded as circuits. They do this by first constructing a circuit that computes the clause-literal incidence matrix of a formula $F(x)$ (*i.e.*, given a clause and a literal, the circuit decides whether the literal occurs in the clause in $F(x)$) such that $F(x)$ is satifiable if and only if $x \in \mathcal{L}(U_{\mathbf{NEXP}})$. Then, using the properties of projection, they construct a circuit representing a graph $G(x)$ such that $G(x)$ has a property π if and only if $x \in \mathcal{L}(U_{\mathbf{NEXP}})$.

When graphs are represented by OBDDs, such reductions are not immediately obtainable, for two basic reasons. First, OBDDs are strictly less powerful than Boolean circuits, in the sense that there are Boolean functions that have small circuit representations but no small OBDD representations. In particular, the function that computes the i^{th} bit of the product (or quotient) of two binary numbers cannot be represented by a small OBDD. Second, the size of OBDDs is sensitive to the ordering of the variables of the function, and the OBDD representing the Boolean combination of two OBDDs can be constructed quickly only when the ordering of the variables is consistent in the two OBDDs. Hence, for a result equivalent to Papadimitriou and Yannakakis's to hold for graphs represented by OBDDs, we must construct reductions f such that the j^{th} bit of $f(x)$ can be found by a small OBDD given j as the input, assuming that the i^{th} bit of x can be found by a small OBDD given i as the input. Furthermore, all OBDDs involved must read the bits in consistent order.

4 A Small OBDD for a NEXP Tableau

Applying Cook's theorem to the tableau of a nondeterministic, exponential-time Turing machine produces a Boolean formula with exponentially many clauses and literals such that the formula is satisfiable if and only if the tableau represents a valid, accepting computation. In this section, we show that this formula can be represented succinctly by an OBDD. This means that we can fix an enumeration of the clauses and literals such that, given the indices of a clause and of a literal, we can determine whether the literal occurs in the clause. Our proof exploits the great regularity of the formula in question. Roughly, we use the fact that, for a given input x, there is a small, fixed constant c such that a literal with index l occurs only in clauses with indices between $(l-1)c+1+K$ and $lc+K$, where K is some (not necessarily small) number.

We begin by proving that this range check can be computed by a small OBDD using a fixed ordering consistent with the one in which the bits of the literal index and the bits of the clause index are interleaved starting from the most significant bit of each.

Lemma 10. *There is a circuit with $w_f = \log Y + 2$ and $w_r = 0$ that, given an ordering $x^{(n)} < x^{(n-1)} < \cdots < x^{(1)}$ on input bits, computes the i^{th} bit of*

x/Y, for a fixed Y and any i, where "/" denotes integer division. Similarly, there is a circuit with $w_f = \log Y + 2$ and $w_r = 0$ that, given an ordering $x^{(n)} < x^{(n-1)} < \cdots < x^{(1)}$ on input bits, computes the i^{th} bit of x mod Y, for a fixed Y and any i.

Proof. The long-division circuit of figure 1, which computes both quotient and remainder, has these properties. □

Lemma 11. *Let f be a function such that $f(x)^{(i)}$ depends only on $x^{(n)}, x^{(n-1)}$, ..., $x^{(i)}$. Given an ordering $x^{(n)} < y^{(n)} < x^{(n-1)} < \cdots < y^{(1)}$ on input bits, the circuit that checks whether $f(x + K) = y$, for a fixed K, has $w_r = 0$ and $w_f = 2W + 4$, where W is the forward width of the circuit that computes $f(x)$ given the same ordering on input bits.*

Proof. The difficulty arises in computing $x + K$, using the given ordering on the input bits. If we used the reverse ordering, going from least significant bits to most significant bits, the OBDD would be simple. Unfortunately, this ordering would make the proof of Lemma 10 very difficult.

In order to compute the i^{th} bit of the sum, we need to know whether $(x^{(i-1)}x^{(i-2)} \cdots x^{(1)} + K^{(i-1)}K^{(i-2)} \cdots K^{(1)})$ yields a carry, and we cannot know this until we compute the sum.

We overcome this obstacle using essentially the same idea as is used in carry-look-ahead adders. A bit position i is a carry *generator* if $x^{(i)}$ and $K^{(i)}$ are 1; it is a carry *propagator* if exactly one of these bits is 1; it is a carry *killer* if both of these bits are 0.

Initially, we compute $f(x)^{(n)}$ in two ways, one assuming that there is a carry into the n^{th} position and the other assuming that there is no carry into the n^{th} position. We compare each value of $f(x)^{(n)}$ with $y^{(n)}$ terminating any computation path that leads to inequality. Inductively, suppose that we have computed at most two different values for $f(x)^{(i)}$ and compared both with $y^{(i)}$. If position $i-1$ is a generator, we abandon the path in which there is no carry into position i and continue the path in which there is a carry into position i in two ways — one assuming there is a carry into position $i-1$ and the other assuming not. Similarly, if position $i-1$ is a propagator, we continue the path assuming a carry into position i by assuming that there is a carry into position $i-1$ and the path assuming that there is no carry into position i by assuming that there is no carry into position $i-1$. Finally, if position $i-1$ is a carry killer, we abandon the path assuming that there is a carry into position i and once more we are left with at most two paths to continue. Clearly, only one of these two paths will be correct at the end, and if along this path we have determined that the bits of $f(x+K)$ are equal to the bits of y, we output a 1, otherwise we output a 0. Our computation is an appropriate interleaving of the carry-look-ahead adder, the circuit that computes f hypothesized in the lemma, and a circuit that checks equality of the bits of $f(x + K)$ and the bits of y. □

Notation: Consider the language $\mathcal{L}(U_{\mathbf{NEXP}})$. Let $F(x)$ be the CNF Boolean formula obtained by the exponential version of Cook's construction, in which $F(x)$ is satisfiable if and only if $x \in \mathcal{L}(U_{\mathbf{NEXP}})$.

Theorem 12. *Let g_x be the Boolean function that decides whether a given literal occurs in a given clause in $F(x)$, i.e., $g_x(Cl, Lt) = 1$ if and only if the literal whose index is Lt occurs in the clause of $F(x)$ whose index is Cl. There is an OBDD of size polynomial in the length of x that represents the function g_x.*

Proof. There are 4 categories of clauses in $F(x)$.

(A) Clauses that state that, at time 0, the tape contains the input string, and the machine is in the initial state.
(B) Clauses that state that, at time $2^{p_M(n)}$, the machine is in one of the final states.
(C) Clauses that state that, at time i, $0 \leq i \leq 2^{p_M(n)}$, each tape cell contains exactly one symbol of the tape alphabet.
(D) Clauses that state that, at time i, $0 \leq i \leq 2^{p_M(n)}$, the contents of the tape cells and the state of the machine follow from those at the previous time $i-1$ by a valid "move" of the machine.

We shall first list all the clauses in category (A), then those in category (B), and then, finally, those in categories (C) and (D). The clauses in categories (C) and (D) will be interleaved so that all clauses in these two categories referring to the same cell of the tableau occur together in the enumeration starting from $\langle 0, 0\rangle$, *i.e.*, cell 0 and time 0, and proceeding in row major order.

An important property that will become clear in the proof is that there is an integer W such that, for each pair i, j, the number of clauses in category (C) and (D) for time i and tape cell j is W, and W depends only on the alphabet size and the maximum nondeterministic branching possible at any state. Hence, for the above listing of the clauses, determining the time and cell to which a given clause number refers involves dividing by this fixed constant W and not by a number that is a function i or j. This is very important, because the function that determines the quotient when one number is divided by another does not, in general, have a small OBDD representation. Because there is a fixed constant W, we can use lemma 10.

When describing the machine at some instant, we will group the state of the machine with the symbol scanned to form a single composite symbol whose appearance also indicates the head position. For each time instant i, for each tape cell j, and for each symbol X, which can either be a symbol of the tape alphabet or a composite symbol, we create a Boolean variable $V_{i,j,X}$ to indicate that the contents of cell j at time i is X. The literals are numbered in the following order: $V_{0,0,\sigma_1}, \neg V_{0,0,\sigma_1}, \ldots, V_{0,0,\sigma_m}, \neg V_{0,0,\sigma_m}, V_{0,1,\sigma_1}, \ldots, \neg V_{0,2^{p_M(n)},\sigma_m}, V_{1,0,\sigma_1}, \ldots, \neg V_{2^{p_M(n)},2^{p_M(n)},\sigma_m}$, where $\Gamma = \{\sigma_1, \sigma_2, \ldots, \sigma_m\}$ is the set of tape symbols and composite symbols.

Notation: We will use $\#(X)$ to denote the index of X, where X is a literal, clause, or symbol in the enumeration.

We will now show that, for the set of clauses in any category, g_x when restricted to this set of clauses can be represented by an OBDD of polynomial size.

Since g_x is a logical **OR** of all these (constant number of) Boolean functions, theorem 6 implies that there is a polynomial-sized OBDD that represents g_x.

In the rest of the proof, we will only consider OBDDs with the following ordering on the variables — $Cl^{(k)}, Lt^{(k)}, Cl^{(k-1)}, Lt^{(k-1)}, \ldots, Cl^{(1)}, Lt^{(1)}$, where Cl is the index of a clause and Lt is the index of a literal.

Case (A): Each clause in this category consists of a single literal. Clauses corresponding to cell positions $\langle 0,0\rangle$ to $\langle 0, n+1\rangle$ specify that the top row faithfully represents the input string $y^{(1)}, \ldots y^{(n)}$ along with suitable end-markers and the start state, while clauses corresponding to cell positions $\langle 0, n+2\rangle, \ldots, \langle 0, 2^{p_M(n)}\rangle$ specify that these symbols are B, the blank symbol.

Note that the OBDD is allowed to be of size polynomial in n. Thus we can compute g_x in the case that the clause number is between 0 and $n+1$ by using an OBDD that resembles a trie. Upon reading a symbol, simply branch to the node in the trie that represents all possible continuations of the clause and literal indices that would make g_x evaluate to 1. The size of the trie (and hence also the size of the OBDD) is $O(n)$.

For clauses whose number is between $n+2$ and $2^{p_M(n)}$, check that $\#(Lt)$ mod $2m$ produces a number that encodes the blank symbol. Lemma 10 implies that this can be done by a small OBDD.

The OR of these two OBDDs computes g_x in case (A).

Case (B): The clauses in this category are

$$\bigvee_j \Big(\bigvee_{X \in F} V_{2^{p_M(n)}, j, X} \Big),$$

where F is the set of composite symbols that encode a final state and symbol pair.

Here we need to test whether $Cl = 2^{p_M(n)} + 1$, $Lt/2m \geq K$, and $Lt \ mod \ 2m$ is a symbol that encodes a final state. Here K is $\langle 2^{p_M(n)}, 0\rangle$. The first two tests clearly have small OBDDs. Because $|F| \leq m$ (the size of the set of tape symbols and composite symbols), the OBDD that decides whether a number ($\leq 2m$) is a symbol in F has at most m paths and so is small. Thus we also have a small OBDD representation for g_x in this case.

Case (C): The clauses in this case are

$\bigvee_X V_{i,j,X}$

$\neg V_{i,j,X} \vee \neg V_{i,j,Y}$, where $X \neq Y$.

Recall that we had interleaved the clauses in category (C) and (D). Thus part of the tests in both these categories will involve checking whether the "block" (*i.e.*, the $\langle i, j\rangle$ pair) is "correct."

The "block," and "offset" within a "block," for the index of a clause can be determined as follows:

$$(Cl - K_1)/K_2 = \text{"block," and}$$

$$(Cl - K_1) \ mod \ K_2 = \text{"offset,"}$$

where K_1 = number of clauses in category (A) and (B) = $2^{p_M(n)} + 1$, and
K_2 = number of clauses in category (C) and (D) for a fixed i and j.

Because K_2 is a constant that depends only on the alphabet size and the maximum nondeterministic branching possible in M, lemmas 10 and 11 and theorem 9 imply that the OBDDs that compute the "block" and "offset" are small.

Subcase 1: The case of $(Cl - K_1) \bmod K_2 = 1$. (Here we are checking that at least one symbol occurs in each cell position.) In this case, we see whether $(Cl - K_1)/K_2 = Lt/2m$, $(Cl - K_1)\ mod\ K_2 = 1$, and $Lt\ mod\ 2m$ is odd (*i.e.*, literal is positive). Each of these tests has a small OBDD, and so Subcase 1 presents no problems.

Subcase 2: The case of $1 < (Cl - K_1)/K_2 \bmod K_2 \leq \binom{m}{2} + 1$. Here we test whether $(Cl - K_1)/K_2 = Lt/2m$ and $Lt\ mod\ 2m$ occurs in the clause whose "offset" is $(Cl - K_1)\ mod\ K_2$.

For each value of $(Cl-K_1)\ mod\ K_2$, the OBDD that checks for the occurrence of $Lt\ mod\ 2m$ has two paths, because each such clause has two literals, and thus it is small. The desired OBDD will have $O(m^2)$ paths, where, in the i^{th} path, one makes the checks for the case $(Cl-K_1)\ mod\ K_2 = i$. The OBDD in Subcase 2 is thus polynomial sized.

Case (D): The Boolean formula that states that any transition of the machine from one configuration to another must follow from a valid "move" of the machine looks like

$$\bigvee_{(W,X,Y,Z)\in Quad} (V_{i,j-1,W} \wedge V_{i,j,X} \wedge V_{i,j+1,Y} \wedge V_{i+1,j,Z}),$$

where $Quad$ is the set of all quadruples (W, X, Y, Z) with the following property: If the symbols in the $(j-1)^{st}$, j^{th}, and $(j+1)^{st}$ cells of the tape are W, X, and Y, respectively, then at the next time instant the symbol at the j^{th} cell will be Z.

Distributing the $\vee$ over the $\wedge$s gives the following set of clauses.

$$V_{i,j-1,W_1} \vee V_{i,j-1,W_2} \vee \cdots \vee V_{i,j-1,W_k},$$

$$V_{i,j-1,W_1} \vee V_{i,j-1,W_2} \vee \cdots \vee V_{i,j,X_1},$$

$$\vdots$$

$$V_{i+1,j,Z_1} \vee V_{i+1,j,Z_2} \vee \cdots \vee V_{i+1,j,Z_k},$$

where $|Quad| = k$.

Subcase 1: The case of $(Cl - K_1)/K_2 = Lt/2m + 1$, *i.e.*, the one in which the literal refers to the $(j-1)^{th}$ cell and time i (or $\langle i, j-1\rangle$). Here, we must check whether the symbol encoded by Lt is one of the W_is occurring in the clause. For

any value of $(Cl - K_1)$ *mod* K_2, the OBDD that checks whether $Lt \bmod 2m$ is one of the W_is occurring in it has at most k paths, because there are at the most k different W_is in a clause. Because $k < |x|$, this OBDD is sufficiently small. Now $(Cl - K_1)$ *mod* K_2 can take at the most $\binom{k+4}{k}$ different values. Thus once again the OBDD is small.

Subcases (2) through (4) are listed below. Arguments showing that there are small OBDDs for these cases are similar to the one in subcase (1).

Subcase 2: $(Cl - K_1)/K_2 = Lt/2m$.

Subcase 3: $(Cl - K_1)/K_2 = Lt/2m - 1$.

Subcase 4: $(Cl - K_1)/K_2 = Lt/2m - 2^{p_M(n)}$.

□

5 NP-Complete Graph Problems

Theorem 12 can be used to prove that most classical NP-complete graph problems are NEXP-complete when graphs are represented by OBDDs. We give one example of such a proof; others are quite similar.

Theorem 13. *The INDEPENDENT SET problem for graphs represented by OBDDs is NEXP-complete.*

Proof. Consider the standard reduction from 3-SAT to INDEPENDENT SET. In this reduction, we create a graph in which there is a node for each occurrence of each literal and two nodes are adjacent if and only if they either correspond to two literals in the same clause or correspond to two complementary literals x and $\bar{x}$ occurring in two different clauses. Let $G_{F(x)}$ be the graph obtained by such a reduction from the formula $F(x)$, for some string x that encodes a $2^{p_M(n)}$-time bounded Turing machine M and an input y. Let each node of $G_{F(x)}$ be named by the clause-literal pair corresponding to it.

Claim: Given two vertices (Cl_1, Lt_1) and (Cl_2, Lt_2) of graph $G_{F(x)}$, there is a polynomial-sized OBDD that decides whether these vertices are adjacent.

Proof. In order to check whether (Cl_1, Lt_1) is adjacent to (Cl_2, Lt_2), we will

(a) Check whether (Cl_1, Lt_1) and (Cl_2, Lt_2) are "valid" vertices in $G_{F(x)}$. If either one of these vertices is not "valid," we will declare them to be adjacent.
(b) If both are "valid," we check whether they are adjacent as per the reduction described above.

A node (Cl, Lt) is "valid" if and only if the literal whose index is Lt occurs in the clause Cl, *i.e.*, if and only if $g_x(Cl, Lt) = 1$. From the previous section, we know that there is a small OBDD that computes g_x, and so we have a small OBDD to check whether a node is valid.

¿From the construction of $G_{F(x)}$, we know that two valid vertices are adjacent if and only if either they correspond to literals in the same clause or they correspond to complementary literals in different clauses, *i.e.*, either $Cl_1 = Cl_2$ or $|Lt_1 - Lt_2| = 1$ and $(Lt_1 - 1)/2 = (Lt_2 - 1)/2$. By lemma 11, both of these checks can be done by small OBDDs whose orderings of the variables are consistent with that of the OBDD for g_x.

□

Thus the adjacency relation for graph $G_{F(x)}$ can be represented by a small OBDD. Furthermore this OBDD can be constructed in polynomial time. $G_{F(x)}$ has an independent set of size K, where K is the number of clauses in $F(x)$, if and only if $F(x)$ is satisfiable if and only if $x \in \mathcal{L}(U_{\mathbf{NEXP}})$.

Hence, the INDEPENDENT SET problem for graphs represented by OBDDs is NEXP-complete.

□

Papadimitriou and Yannakakis [18] prove the general theorem that, if the reduction from SAT to an NP-complete problem π is a projection, then π becomes NEXP-complete when the input is represented by a circuit. The fact that the reduction is a projection is a sufficient condition for their result, but it appears far from necessary.

Here we state an analogous result. Let f be a reduction from SAT to an NP-complete problem π. Suppose there is a constant k such that, for all j, $f(x)^{(j)}$ is a function only of the bits $x^{(j_1)}, \ldots, x^{(j_k)}$. Moreover, suppose that there is a finite automaton similar to a Mealy machine that takes the bits of j in some canonical order as input and produces the bits of $j_1, \ldots, j_k$ in most-significant to least-significant order. We will refer to the above class of reductions as NC^0-padding reductions. Note that the class of NC^0-padding reductions is incomparable with the class of projections.

We state the following theorem. The proof is omitted.

Theorem 14. *Let f be an NC^0-padding reduction from SAT to a problem π in NP. Then π is NEXP-complete when its instances are presented as OBDDs.*

NC^0-padding reductions can be found for a number of graph problems such as CLIQUE, VERTEX COVER, *etc.* Such reductions are obtained by taking the standard reductions and padding the target instances so that each of its indices has sufficient information to allow the reconstruction of the indices on which it depends.

6 The GAP and AGAP Problems

In this section, we examine two graph problems that are crucially important in computer-aided verification. We show that both experience exponential blow-up in worst-case complexity when instances are represented as OBDDs.

Problem 15. (GAP) The Graph Accessibility Problem is:
Input A directed graph G and vertices s and t in the graph.
Output Is there a directed path from s to t ?

Theorem 16. *GAP is PSPACE-complete when the graph G is represented by an OBDD.*

Proof. Let p be a polynomial and x be a string that encodes a $p_M(|y|)$-space-bounded Turing machine M and an input y. Without loss of generality, we may assume that the machine M has a single accepting configuration C_f.

Consider the configuration graph $G_{M,y} = (V_{M,y}, E_{M,y})$ corresponding to the machine M and input y, where

$V_{M,y} = \{v_C \mid C$ is a possible configuration of machine *i.e.*, C is a string of symbols, of which one is a composite symbol encoding a state of machine M and a tape symbol, and all the rest are tape symbols $\}$, and
$E_{M,y} = \{< v_{C_1}, v_{C_2} > \mid$ the machine M can go from configuration C_1 to a configuration C_2 in one step $\}$.

If C_i is the initial configuration of machine M on input y, then M accepts y if and only if the node labeled by C_f is reachable from the node labeled by C_i in $G_{M,y}$. In other words, $x \in \mathcal{L}(U_{\mathbf{PSPACE}})$ if and only if the GAP problem on $G_{M,y}$ has the answer "yes."

Claim: The edge relation $E_{M,y}$ in graph $G_{M,y}$ has a small OBDD representation.

Proof. We need to show that the function e, where,

$$e(C_1, C_2) = \begin{cases} 1 \text{ if and only if } < v_{C_1}, v_{C_2} > \in E_{M,y} \\ 0 \text{ otherwise} \end{cases}$$

can be represented by a small OBDD. Computing the function e entails:

(a) checking that C_1 and C_2 are possible configurations, *i.e.*, there is exactly one composite symbol in each of C_1 and C_2, and
(b) checking whether the configuration C_2 can be reached from C_1 in one step by the machine M.

Checking to see whether a symbol is composite amounts to checking whether the index of the symbol is greater than some constant, because we list all the composite symbols in the end. Hence, check (a) only involves examining the symbols of the configuration in the order in which they occur and thus has a small OBDD representation.

Let $Quad = \{(W, X, Y, Z) |$ if W, X, and Y are the symbols in the $(j-1)^{st}$, j^{th} and $(j+1)^{st}$ cells, respectively, at some time instant, then Z is the symbol in the j^{th} cell at the next time instant $\}$. Checking whether configuration C_2 can be reached from configuration C_1 in one step involves checking whether all

the symbols in C_2 arise from the corresponding symbols in C_1. That is, we need to check that $\forall j, (C_1^{(j-1)}, C_1^{(j)}, C_1^{(j+1)}, C_2^{(j)}) \in Quad$. As we saw in the proof of theorem 12, the function that checks whether a given quadruple is in $Quad$ can be represented by a small OBDD. We just read the symbols of C_1 and C_2 alternately and keep checking whether they "conform." Note that we need to "remember" only two symbols of C_1 as we go along. Hence, at any level in the OBDD, there are at most a constant number of nodes, and checking whether one configuration can follow from another is representable by a small OBDD.

□

Because $G_{M,y}$ can be represented by a small OBDD that can be constructed in polynomial time, the GAP problem for graphs represented by OBDDs is PSPACE-complete.

□

Definition 17. An AND-OR graph is a directed graph G with vertices labeled "AND" or "OR." Reachability in such graphs is recursively defined as follows :

(a) Every vertex is reachable from itself.
(b) If u is an AND node, then v is reachable from u if and only if v is reachable from **all** u_i, such that $< u, u_i >$ is an edge in the graph.
(c) If u is an OR node, then v is reachable from u if and only if v is reachable from **any** u_i, such that $< u, u_i >$ is an edge in the graph.

Problem 18. (AGAP)
The Alternating Graph Accessibility Problem is:
Input An AND-OR graph G and vertices s and t in G.
Output Is t reachable from s ?

Theorem 19. *The AGAP problem for graphs represented by OBDDs is EXP-complete.*

Proof. Because the AGAP problem is in P for graphs represented by adjacency matrices, it is in EXP for graphs represented by OBDDs.

Let x be a string that encodes a $2^{p_M(n)}$-time bounded Turing machine M and an input y. We will construct an AND-OR graph with two special vertices s and t, such that t will be reachable from s if and only if $x \in \mathcal{L}(U_{\mathbf{EXP}})$. The construction of the graph is very similar to the construction of the circuit in the proof that CIRCUIT VALUE is P-complete.

Once again let $Quad = \{(W, X, Y, Z) |$ if W, X, and Y are the symbols in the $(j-1)^{st}$, j^{th}, and $(j+1)^{st}$ cells, respectively, at some time instant, then Z is the symbol in the j^{th} cell at the next time instant $\}$. Let $<$ be some ordering on the quadruples in $Quad$.

We will construct the graph $G_{M,y}$ in stages, starting with the empty graph.

Stage 0: Add two AND nodes, one labeled 0 and the other 1. These nodes represent "false" and "true," respectively.

Stage 1: For each $j, 0 \leq j \leq 2^{p_M(n)}$, and each X, where X is either a tape symbol or a composite symbol encoding a state of machine M and a tape symbol, add an OR node labeled $V_{0,j,X}$. Add the edge $< V_{0,j,X}, 1 >$ if the j^{th} symbol in the initial configuration of M on input y is X. Otherwise, add the edge $< V_{0,j,X}, 0 >$.

Stage 2i: For each j and k, add an AND node labeled $N_{i,j,k}$. Add edges $< N_{i,j,k}, V_{i-1,j-1,W} >, < N_{i,j,k}, V_{i-1,j,X} >$, and $< N_{i,j,k}, V_{i-1,j-1,Y} >$, where the k^{th} quadruple in $Quad$ is (W, X, Y, Z), for some Z.

Stage 2i+1: For each j and symbol Z, add an OR node labeled $V_{i,j,Z}$. For each k, if (W, X, Y, Z) is the k^{th} quadruple, for some W, X, and Y, then add the edge $< V_{i,j,Z}, N_{i,j,k} >$.

Stage $2^{p_M(n)} + 2$: Add an OR node s. For all j, add edges $< s, V_{2^{p_M(n)},j,X} >$, where X is a composite symbol encoding a final state and some tape symbol.

The basic idea of the construction is as follows. The node-label $V_{i,j,X}$ means that, during the computation, at time i, the j^{th} tape cell contains the symbol X. From the definition of $Quad$, it can be seen that

$$V_{i,j,Z} = \bigvee_{(W,X,Y,Z) \in Quad} (V_{i-1,j-1,W} \wedge V_{i-1,j,X} \wedge V_{i-1,j+1,Y}).$$

The string y is accepted if, at time $2^{p_M(n)}$, the machine reaches a final state, *i.e.*,

$$\bigvee_j (\bigvee_{X \in F} V_{2^{p_M(n)},j,X}),$$

where F is the set of composite symbols that encode a final state and symbol pair. Hence, the graph $G_{M,y}$ is such that node 1 is reachable from node s if and only if M accepts input y.

Claim: The graph $G_{M,y}$ can be represented by a small OBDD.

Proof. The only interesting edges are those of the form $< N_{i,j,k}, V_{i-1,j',X} >$, where $j' = j$ or $j - 1$ or $j + 1$, and $< V_{i,j,X}, N_{i,j,k} >$. In each case, determining whether nodes $V_{i_1,j_1,X}$ and $N_{I_2,j_2,k}$ are adjacent involves checking whether i_1, i_2 and j_1, j_2 differ by a constant and whether the symbol X occurs in the k^{th} quadruple. That both these checks can be done in by a small OBDD was seen in the proof of theorem 12.

□

Thus the graph $G_{M,y}$ can be represented by a small OBDD that can be constructed in polynomial time. Furthermore, node 1 is reachable from s in $G_{M,y}$ if and only if $x \in \mathcal{L}(U_{\mathbf{EXP}})$. Hence, the AGAP problem is EXP-complete for graphs represented by OBDDs.

□

7 Open Questions

The results that we prove in this paper are all negative; they show that, in the worst case, succinct encoding of instances using OBDDs results in problems that are hard for PSPACE, EXP, or NEXP. However, one of our main motivations for this investigation is the observed good performance of computer-aided verification tools on OBDD-encoded instances. Thus worst-case hardness results do not adequately capture the complexity of the problems on "real-world instances," as is often the case. It would be desirable to have precise characterizations of the special cases that occur in practice and of the special cases that can be solved efficiently.

It would also be desirable to have a general hardness result for OBDDs that is analogous to Papadmitriou and Yannakakis's result for circuits. In other words, is there a class of reductions such that any problem that is complete for NP via a reduction in the class is complete for NEXP when instances are encoded as OBDDs?

8 Acknowledgement

It is a pleasure to thank Ed Clarke for helpful comments on an earlier draft of this paper.

References

1. J. L. Balcázar, *The Complexity of Searching Implicit Graphs*, Aritificial Intelligence, 86 (1996), pp. 171-188.
2. J. L. Balcázar and A. Lozano, *The complexity of graph problems for succinctly represented graphs*, in Proc. Graph-Theoretic Concepts in Computer Science, Springer-Verlag, Lecture Notes in Computer Science 411, 1989, pp. 277-285.
3. J. L. Balcázar, A. Lozano, and J. Torán, *The complexity of Algorithmic Problems on Succinct Instances*, Computer Science (R. Baeza-Yates and U. Manber, Eds.), Plenum Press, New York, 1992, pp. 351-377.
4. I. Beer, S. Ben-David, D. Geist, R. Gewirtzman, and M. Yoeli, *Methodology and system for practical formal verification of reactive hardware*, in Proc. 6th Int'l Conf. on Computer-Aided Verification, Springer-Verlag, Lecture Notes in Computer Science 818, 1994, pp. 182–193.
5. C. L. Berman, *Ordered Binary Decision Diagrams and Circuit Structure*, in Proc. IEEE International Conf. on Computer Design, 1989, pp. 392–395.,
6. R. Brayton, A. Emerson, and J. Feigenbaum, *Workshop Summary: Computational and Complexity Issues in Automated Verification*, DIMACS Technical Report 96-15, Rutgers University, Piscataway NJ, June 1996.
7. R. Bryant, *Graph-Based Algorithms for Boolean Function Manipulation*, IEEE Transactions on Computers, C-35 (1986), pp. 677–691.
8. R. Bryant, *Symbolic Manipulation with Ordered Binary Decision Diagrams*, ACM Computing Surveys, 24 (1992), pp. 293–318.

9. J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang, *Symbolic model checking:* 10^{20} *states and beyond*, Information and Computation, 98 (1992), pp. 142–170.
10. E. M. Clarke, O. Grumberg, H. Hiraishi, S. Jha, D. E. Long, K. L. McMillan, and L. A. Ness, *Verification of the Futurebus+ Cache Coherence Protocol*, Formal Methods in System Design, 6 (1995), pp. 217–232.
11. T. Cormen, C. Leiserson, and R. Rivest, *Introduction to Algorithms*, The M.I.T. Press, Cambridge, MA (1989).
12. E. A. Emerson and C. L. Lei, *Efficient Model Checking in Fragments of the Propositional mu-Calculus*, Proc. 1st IEEE Symp. on Logic in Computer Science, 1986, pp. 267–278.
13. H. Galperin and A. Wigderson, *Succinct Representations of Graphs*, Information and Control, 56 (1983), pp. 183–198.
14. O. Kupferman and M. Y. Vardi, *Module checking*, Proc. 8th Int'l. Conf. on Computer-Aided Verification, Springer-Verlag, Lecture Notes in Computer Science 1102, 1996, pp. 75–86.
15. R. Kurshan, *Computer-Aided Verification of Coordinating Processes: The Automata-Theoretic Approach*, Princeton University Press, Princeton NJ, 1994.
16. R. Kurshan, *The Complexity of Verification*, Proc. 26th ACM Symp. on Theory of Computing, Montreal, 1994, pp. 365–371.
17. K. McMillan, *Symbolic Model Checking*, Kluwer Academic Publishers, Amsterdam, 1993.
18. C. Papadimitriou and M. Yannakakis, *A Note on Succinct Representations of Graphs*, Information and Control, 71 (1986), pp. 181–185.
19. S. Ponzio, *A lower bound for integer multiplication with read-once branching programs*, Proc. 27th ACM Symp. on Theory of Computation, Las Vegas, 1995, pp. 130–139.
20. S. Ponzio, *Restricted Branching Programs and Hardware Verification*, PhD Thesis, MIT, 1995.
21. S. Skyum and L. Valiant, *A Complexity Theory Based on Boolean Algebra*, in Proc. 23rd IEEE Symp. on Foundations of Computer Science, 1982, pp. 244-253.
22. J. Torán, *Succinct representations of counting problems*, Proc. 6th International Conf. on Applied Algebra, Algebraic Algorithms, and Error-Correcting Codes, Springer-Verlag, Lecture Notes in Computer Science 357, 1988, pp. 415-426.
23. H. Veith, *Languages Represented by Boolean Formulas*, TU Vienna, TR CD 85/95, 1995.
24. H. Veith, *Succinct Representation, Leaf Languages, and Projection Reductions*, in Proc. 11th IEEE Conf. on Computational Complexity, 1996, pp. 118-126.

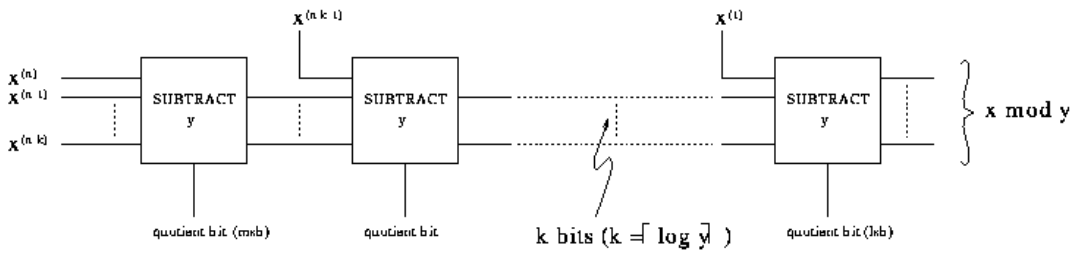

Fig. 1. Circuit that computes the quotient and remainder of two numbers x and y