JOURNAL OF COMPLEXITY **15**, 17–29 (1999)
Article ID jcom.1998.0494, available online at http://www.idealibrary.com on IDEAL®

Lower Bounds on the Depth of Monotone Arithmetic Computations*

Don Coppersmith and Baruch Schieber

IBM—Research Division, T. J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598
E-mail: copper@watson.ibm.com, sbar@watson.ibm.com

Received June 26, 1997

DEDICATED TO ZVI GALIL'S 50TH BIRTHDAY

Consider an arithmetic expression of length n involving only the operations $\{+, \times\}$ and non-negative constants. We prove lower bounds on the depth of any

computation trees of depth at least $1.5 \log_2 n - O(1)$, thus proving a conjecture of S. R. Kosaraju (1986, *in* "Proc. of the 18th ACM Symp. on Theory Computing," pp. 231–239). In contrast, Kosaraju showed how to compute such expressions with computation trees of depth $2 \log_2 n + O(1)$.

1. INTRODUCTION

The problem of restructuring an arithmetic expression into an equivalent one of reduced depth is one of the basic problems in parallel computation [BKM73, B74, BM75, KM75, MP76, SS80, JS82, Kos86]. Despite long-term efforts, this problem is still unsettled, even for arithmetic expressions that seem simple to compute. To escape from some of the difficulties of the general problem researchers have considered the restricted *monotone* computation model. We follow this direction and prove lower bounds on the depth of monotone arithmetic expressions.

Consider an arithmetic expression of length n involving only the operations $\{+, \times\}$ and nonnegative constants. (The *length* of an arithmetic

* A preliminary version of this paper appeared in "Proceedings, 33rd Symposium on Foundations of Computer Science," pp. 288–295, 1992.

0885-064X/99 $30.00

expression is the minimum number of nodes in a binary computation tree that computes it, where the minimum is taken over all such computation trees.) We wish to evaluate such an arithmetic expression by a binary computation tree of minimum depth over the same sets of operators and constants. We exhibit a family of arithmetic expressions that requires computation trees of depth at least $1.5 \log_2 n - O(1)$; the corresponding upper bound for this family is $2 \log_2 n + O(1)$.

Brent [B74] showed that any arithmetic expression of length n could be computed by a binary tree of depth $O(\log n)$. This is somewhat surprising, since the computation tree suggested by the form of the original expression might be quite skewed and have depth $\Theta(n)$. Brent, Kuck, and Maruyama [BKM73] showed that any arithmetic expression involving only the operations $\{+, \times\}$ and non-negative constants could be computed by a binary tree of depth $2.465 \log n + O(1)$ (throughout, all logarithms are base 2). Later, Muller and Preparata [MP76] improved the constants and showed that such an arithmetic expression could be computed by a binary tree of depth $2.08 \log n$. On the other hand, Shamir and Snir [SS80] showed a lower bound of $1.16 \log n$ for the family of arithmetic expressions defined by alternating 2-3 trees: trees in which nodes at odd levels multiply two subexpressions, and nodes at even levels add three subexpressions. (Note that these trees are not binary and hence not computation trees.) Kosaraju [Kos86] improved Muller and Preparata's upper bound to $2 \log n$. He also conjectured that expressions requiring depth at least $(1 + \varepsilon) \log n$ could be found among the class of arithmetic expressions where at least one operand of each addition node is a leaf. We will refer to such arithmetic expressions as *leaf-addition expressions*.

We prove Kosaraju's conjecture. Our arithmetic expressions are given by trees in which nodes at odd levels multiply two subexpressions, and nodes at even levels (including the top level) add a subexpression and a leaf. This description gives a binary computation tree of depth $2 \log n + O(1)$ for an arithmetic expression of length n. We show that any binary computation tree computing this arithmetic expression must have depth at least $1.5 \log n - O(1)$.

The monotone computation model does not allow division, subtraction, and negative constants, so that cancellation is not possible. This technicality enables our lower bound proofs to go through. Some arguments can be brought in favor of considering such a restrictive computation model. First, this model has absolute numerical stability [Mil75]. Also, upper bounds generalize to any semiring (i.e., a domain with two binary operations $+$ and $\times$, both associative and commutative, with $\times$ distributing over $+$).

Most of the known lower bounds for monotone (arithmetic) computation consider only the weight (number of monomials in a polynomial) of

a polynomial of a given degree computable by a tree of a given depth [SS80, JS82, Sni91]. To get our $1.5 \log_2 n - O(1)$ lower bound we need to look more closely at the specific structure of our arithmetic expression, as well as the shapes of the subtrees appearing under each multiplication node. This new technique may be found useful for proving other lower bounds.

To get our lower bound, we had to prove a slightly stronger lower bound. We actually show that given a polynomial, any polynomial with the same set of monomials, but with *arbitrary* positive coefficients, also requires depth at least $1.5 \log n - O(1)$. It would be interesting if it turns out that one such polynomial is indeed easier to compute than the given one.

We prove our lower bound for leaf-addition expressions. Interestingly, leaf-multiplication expressions (i.e., expressions given by trees where at least one operand of each multiplication node is a leaf) are very efficient to restructure. Kosaraju [Kos86] gave a tight bound of $\log n + \sqrt{\log n} + o(\sqrt{\log n})$ on the depth of computation trees for such arithmetic expressions. The reason for the difference is that $\times$ distributes over $+$, while $+$ does not distribute over $\times$. Notice that this is not the case in the monotone Boolean case with the operations $\vee$, $\wedge$. Since $\wedge$ distributes over $\vee$ and $\vee$ distributes over $\wedge$, we can get a tight bound of $\log n + \sqrt{\log n} + o(\sqrt{\log n})$ on the depth of computation trees for both leaf-addition and leaf-multiplication *Boolean* expressions.

We remark that the bounds for monotone *Boolean* expressions are different. On the positive side, Preparata and Muller [PM76] and later Preparata, Muller, and Barak [PMB77] showed that any Boolean expression with n literals involving only the operations $\{\vee, \wedge\}$ could be computed by a binary tree of depth $1.81 \log n + O(1)$. However, no non-trivial lower bound on the depth is known for the Boolean monotone case.

The rest of the paper is organized as follows. In Section 2 we define the family of leaf-addition expressions. In Section 3 we prove the lower bound for these expressions. Finally, we list some open problems.

2. LEAF-ADDITION EXPRESSIONS

We define the family of arithmetic expressions for which we prove the lower bound. These arithmetic expressions are given by polynomials, each of which has a *level* k, for some non-negative integer k. The length of the polynomial of level k is $2^{k+2} - 3$, and it involves $2^{k+1} - 1$ indeterminates. We show that the evaluation of a polynomial of level k requires depth at least $1.5k$.

We describe a computation tree T_k of depth $2k$ that computes the polynomial of level k. We start from an alternating 2-2 tree of depth $2k$ (i.e., a complete binary tree of depth $2k$, in which nodes at odd levels multiply two subexpressions, and nodes at even levels add two subexpressions). Notice that the root node is an addition node. Now, for each addition node we prune its right subtree and make it a leaf. It is not difficult to see that the resulting tree T_k has $2^k + 2^k - 1 = 2^{k+1} - 1$ leaves (indeterminates) and $2^{k+1} - 2$ computation nodes.

For the lower bound proof it is more convenient to use an alternative recursive definition of the polynomials. First, we introduce some notations.

Notation. We associate to each indeterminate a *range*: a subscript indicating a half-open interval of non-negative integers, for example, $x_{[2,4)}$. For an interval $[a, b)$, we always have that $b-a$ divides a. The polynomials are indexed by two numbers; the first is its level and the second is the "starting point" of the ranges of its indeterminates. For an index (k, a), we always have that 2^k divides a.

The polynomials we wish to compute are defined recursively:

$$P_{(0,a)} = x_{[a,a+1)} \equiv x_a$$

$$\text{for all } k \text{ s.t. } 2^k \mid a, \qquad P_{(k,a)} = P_{(k-1,a)} \times P_{(k-1,a+2^{k-1})} + x_{[a,a+2^k)}.$$

Thus, for example,

$$P_{(3,0)} = [(x_0x_1 + x_{[0,2)})(x_2x_3 + x_{[2,4)}) + x_{[0,4)}] \times [(x_4x_5 + x_{[4,6)})(x_6x_7 + x_{[6,8)}) + x_{[4,8)}] + x_{[0,8)}.$$

To see that this recursive definition defines the same polynomials as the trees T_k, associate the indeterminates $x_{[j,j+1)}$, for $j = 0, ..., 2^k - 1$, to the leaves of T_k at depth $2k$, and the indeterminates $x_{[j2^i,(j+1)2^i)}$, for $i = 1, ..., k$ and $j = 0, ..., 2^{k-i} - 1$, to the leaves at depth $2k - 2i + 1$. Then, it is not difficult to check that T_k computes $P_{(k,0)}$.

We define a *pseudo-degree* of each such polynomial. The pseudo-degree of a polynomial is a set of vectors of non-negative integers. The pseudo-degree of $x_{[a,b)}$ is the singleton set consisting of a vector of non-negative integers, whose ith component is 1 if $a \leqslant i < b$, and 0 otherwise. The pseudo-degree of a sum of two polynomials is the union of the pseudo-degrees of its summands. The pseudo-degree of a product of two polynomials is the set of vectors given by $\mathbf{V}_1 + \mathbf{V}_2$ (componentwise addition), for all vectors $\mathbf{V}_1$ in the pseudo-degree of the first multiplicand and $\mathbf{V}_2$ in the pseudo-degree of the second multiplicand.

The following propositions follow directly from our definition of pseudo-degree.

PROPOSITION 1. *Each polynomial $P_{(k,a)}$ is pseudo-homogeneous. That is, the pseudo-degree of each such polynomial consists of a singleton set.*

PROPOSITION 2. *The ith component of the pseudo-degree of $P_{(k,a)}$ is* 1 *if $a \leqslant i < a + 2^k$ and* 0 *otherwise. Further, every monomial of this pseudo-degree in the given indeterminates is represented in $P_{(k,a)}$.*

3. THE LOWER BOUND

We start with some definitions.

DEFINITION 1. The *weight* of a computation tree with (maximum) depth i and with a multiplication node at its root is 2^i. The weight of a computation tree with an addition node at its root is the sum of the weights of its two subtrees (the left and the right). A computation tree consisting of a single leaf has weight 1.

DEFINITION 2. Let T be a computation tree with a multiplication node at its root. The two trees rooted at the children of the root are called the *major subtrees* of T, and denoted Left(T) and Right(T).

DEFINITION 3. Define $\tilde{P}_{(k,a)}$ to be any polynomial containing the same monomials as $P_{(k,a)}$, but with arbitrary positive (nonzero) coefficients.

Define $D(k)$ as the minimum possible weight of a tree computing $\tilde{P}_{(k,a)}$ for any a. (This weight is independent of a.)

THEOREM 4. *If $k = 2j$ is even, then $D(k) > 2^{3j-1}$. If $k = 2j+1$ is odd, then $D(k) > 3 \cdot 2^{3j-1}$.*

It is easy to see that the depth of a tree of weight w is at least $\log w$. Thus we have the following corollary.

COROLLARY 5. *There are polynomials P of length n involving the operations $\{+, \times\}$ and non-negative constants, such that any binary computation tree over the same set of operations and constants computing P has depth at least* $1.5 \log_2 n - O(1)$.

Proof of Theorem 4. The proof is by induction on k. Clearly, $D(0) = 1 > \frac{1}{2}$ and $D(1) = 3 > \frac{3}{2}$.

Suppose the first violation comes at $k = 2j$, that is, $D(2j) \leqslant 2^{3j-1}$ but $D(2j-1) > 3 \cdot 2^{3j-4}$. Consider a computation tree T for some $\tilde{P} \equiv \tilde{P}_{(k,0)}$ with weight at most $D \equiv 2^{3j-1}$. The proof is by contradiction: we identify three

different subtrees of T each of which computes some $\tilde{P}_{(2j-1,a)}$ and show that the weight of one of them is at most $\frac{3}{8}D = 3 \cdot 2^{3j-4}$, a contradiction.

We find it useful to denote two specific indeterminates

$$L = x_{[0,2^{k-1})}, \qquad R = x_{[2^{k-1},2^k)}.$$

Repeatedly, separate the tree if it has an addition at its root, so that we have expressed $\tilde{P}$ as a sum of polynomials, each of which is computed as a single node or as a tree with a multiplication at its root. The sum of the weights of these trees is at most D.

These trees fall into five classes:

1. N_{LR} is the class of those trees that contain both indeterminates L and R as leaves.
2. N_L is the class of those trees that contain L but not R as a leaf.
3. N_R is the class of those trees that contain R but not L as a leaf.
4. One tree of weight one consisting of the leaf $x_{[0,2^k)}$.
5. N_X is the class of those trees that do not contain any of L, R, or $x_{[0,2^k)}$ as leaves.

Denote by $W(N)$ the sum of the weights of the trees in class N. We have

$$D = 2^{3j-1} \geqslant W(N_{LR}) + W(N_L) + W(N_R) + W(N_X) + 1$$
$$D = 2^{3j-1} > W(N_{LR}) + W(N_L) + W(N_R) + W(N_X).$$

In the following lemmas we prove some properties of the tree T. Without loss of generality we assume that T does not contain multiplications by zero.

LEMMA 6. *Any polynomial computed in the tree T is a pseudo-homogeneous polynomial* (*under our definition of pseudo-degree*).

Proof. Since cancellation is not allowed (and T does not contain multiplications by zero), an inhomogeneous polynomial multiplied by or added to another polynomial will yield an inhomogeneous polynomial. But the final $\tilde{P}$ is pseudo-homogeneous. ▮

Since all the computed polynomials are pseudo-homogeneous, we refer to their pseudo-degree as a vector (rather than a singleton set consisting of a vector).

DEFINITION 7. The *pseudo-degree* of a subtree of T is the pseudo-degree of the (pseudo-homogeneous) polynomial that this subtree computes.

LEMMA 8. *Let P and Q be two polynomials whose pseudo-degrees have a common positive component. Then, P and Q may not be multiplied in T.*

Proof. The pseudo-degree of the resulting polynomial would have a component that is at least 2. Since (multiplicative) cancellation is not allowed the value of this component cannot be lowered in subsequent computation. However, the pseudo-degree of $\tilde{P}$ has components at most 1. ∎

We conclude that the pseudo-degrees of all computed polynomials are zero-one vectors.

LEMMA 9. *There is a computation tree of minimal weight that does not use constants.*

Proof. If zero is added, the zero and the addition node can be deleted. If a positive constant is added to a nonconstant polynomial, we produce an inhomogeneous intermediate result. If zero is multiplied, the whole subtree can be deleted. If a positive constant is multiplied, the constant and the multiplication node can be deleted, and only the constants in $\tilde{P}$ will be affected (but will remain positive). In any case we still compute some $\tilde{P}_{(k,0)}$ without increasing the weight. ∎

Henceforth we assume that our tree has no constants.

LEMMA 10. *For each tree S in N_{LR} the root is a multiplication with one factor having the same pseudo-degree as L (i.e., a vector whose ith component is 1 for $0 \leqslant i < 2^{k-1}$ and 0 otherwise), and the other factor having the same degree as R.*

Proof. Without loss of generality assume that the leaf L appears in Left(S) (and possibly in Right(S) as well). This implies that the pseudo-degree of Left(S) is at least the pseudo-degree of L. (The partial order relation is the componentwise order.) Since the pseudo-degrees of the two major subtrees cannot have a common positive component, the pseudo-degree of Right(S) is at most the pseudo-degree of R. At least one major subtree contains R, and its pseudo-degree is at least the pseudo-degree of R. If the same major subtree Left(S) contains both L and R, then the pseudo-degree of Left(S) is at least the pseudo-degree(L) + pseudo-degree(R) = pseudo-degree($\tilde{P}$), so that the pseudo-degree of Right(S) = 0, and Right(S) computes a constant; by minimality, this does not happen. So L appears only in Left(S), and R appears only in Right(S). This implies that the pseudo-degrees of the two major subtrees must be exactly pseudo-degree(L) and pseudo-degree(R), respectively. ∎

Now, we identify two subtrees of T that compute $\tilde{P}_{(k-1,0)}$. The first one is given by the cofactor of R in $\tilde{P}$ (i.e., the factors that multiply R in all the monomials of $\tilde{P}$ that contain R). Note that by definition this cofactor is

some $\tilde{P}_{(k-1,0)}$. The monomials of the cofactor are contained in N_R and the left major subtrees of N_{LR}. Note that the depth of a left major subtree is one less than the depth of its originating tree. Thus we have

$$\tfrac{1}{2}W(N_{LR}) + W(N_R) \geqslant D(k-1) > 3 \cdot 2^{3j-4} = \tfrac{3}{8}D. \tag{1}$$

(The latter inequality comes from the induction hypothesis.) Similarly, the cofactor of L in $\tilde{P}$ is some $\tilde{P}_{(k-1,2^{k-1})}$, which is equivalent to $\tilde{P}_{(k-1,0)}$. Thus we also have

$$\tfrac{1}{2}W(N_{LR}) + W(N_L) \geqslant D(k-1) > 3 \cdot 2^{3j-4} = \tfrac{3}{8}D. \tag{2}$$

Combining with the estimate on $D(k)$, we get

$$W(N_X) < D(k) - (W(N_{LR}) + W(N_L) + W(N_R)) < \tfrac{1}{4}D. \tag{3}$$

Partition the set of trees in the class N_X into two subsets, $N_X = A_X \oplus B_X$. Each tree in N_X belongs wholly to A_X or wholly to B_X. Beyond this, the partition is *arbitrary*. Similarly, partition N_L, N_R, N_{LR}.

For a monomial M with pseudo-degree(M) $\geqslant$ pseudo-degree(L), let LFact(M) be the factor of M for which pseudo-degree(Lfact(M)) = pseudo-degree(L). Similarly, for a monomial M with pseudo-degree(M) $\geqslant$ pseudo-degree(R), let Rfact(M) be the factor of M for which pseudo-degree(Rfact(M)) = pseudo-degree(R).

To identify the third subtree that computes $\tilde{P}_{(k-1,0)}$ we need the following lemmas.

LEMMA 11. *For each tree in N_{LR} or N_L (resp. N_R) that computes the polynomial $\sum_i \alpha_i M_i$, we can construct a tree with strictly smaller depth that computes the polynomial $\sum_i \alpha_i$ LFact(M_i) (resp. $\sum_i \alpha_i$ RFact(M_i)).*

Proof. We prove only for $\sum_i \alpha_i$ LFact(M_i); the proof for $\sum_i \alpha_i$ RFact(M_i) is analogous. Consider a tree S in $N_{LR} \cup N_L$, and assume without loss of generality that the major subtree containing L is Left(S). Clearly, the pseudo-degree of Left(S) is at least pseudo-degree(L), and the pseudo-degree of Right(S) is no more than pseudo-degree(R). Then, for each M_i, Lfact(M_i) is a factor of a monomial in the polynomial computed by Left(S). Eliminate from Left(S) indeterminates whose pseudo-degree is not part of pseudo-degree(L) to achieve the desired polynomial.

For an arbitrary partition of N_{LR}, N_L, N_R, and N_X, let

$$\mathscr{W}_A = \tfrac{1}{2}W(A_{LR}) + \tfrac{1}{2}W(A_L) + W(A_R) + W(A_X)$$
$$\mathscr{W}_B = \tfrac{1}{2}W(B_{LR}) + W(B_L) + \tfrac{1}{2}W(B_R) + W(B_X).$$

LEMMA 12. *We can construct a tree that computes some* $\tilde{P}_{(k-1,0)}$ *whose weight is at most* $\max\{\mathscr{W}_A, \mathscr{W}_B\}$.

Proof. Consider those monomials M such that pseudo-degree(M) = pseudo-degree(R) and M is a factor of some monomial in $\tilde{P}$. The sum of those monomials is some $\tilde{P}_{(k-1,2^{k-1})}$. Either all such monomials M occur as factors of monomials in the polynomials computed by the trees in $\mathscr{B} \equiv B_{LR} \oplus B_L \oplus B_R \oplus B_X$, or one such monomial M fails to appear there.

Suppose that all the monomials M appear as factors in $\mathscr{B}$. We bound the weight of a tree that computes the sum of these monomials. This tree is given by adding all the trees in $\mathscr{B}$ after eliminating from them all indeterminates whose pseudo-degree is less than or equal to pseudo-degree(L). By Lemma 11 the depth of any tree in $B_{LR} \cup B_R$ after eliminating from it all these indeterminates is strictly less than the depth of its originating tree. Thus the contribution of the trees in $B_{LR} \cup B_R$ to the weight of the tree that computes the sum is at most $\frac{1}{2}(W(B_{LR}) + W(B_R))$. The contribution of the trees in $B_L \cup B_X$ is at most $W(B_L) + W(B_X)$. It follows that the weight of the tree that computes the sum of the monomials M is at most $\mathscr{W}_B$.

If a monomial M fails to appear in $\mathscr{B}$ then its cofactor, which is some $\tilde{P}_{(k-1,0)}$, is computed by adding all the trees in $\mathscr{A}$ after eliminating from them all indeterminates whose pseudo-degree is less than or equal to pseudo-degree(R). Similar to the first case, the weight of this tree is at most $\mathscr{W}_A$. ∎

We would like to minimize the value of $\max\{\mathscr{W}_A, \mathscr{W}_B\}$. For this we consider the weights of possible partitions of each of the tree classes. Let N be one of $\{N_X, N_L, N_R, N_{LR}\}$ and consider the partition $N = A \oplus B$. Clearly, $W(N) = W(A) + W(B)$. Recall that the weight of each tree in N is a power of two. The restriction that each tree in N must belong wholly either to A or to B implies a restriction on the possible values of $W(A)$. In the following lemmas we bound the gaps between achievable values of $W(A)$.

LEMMA 13. *Let w be a non-negative integer whose binary representation is a subset of the binary representation of $W(N)$ (that is, wherever the binary representation of w has a* 1, *the binary representation of $W(N)$ also has a* 1). *Then there is a partition $N = A \oplus B$ with $W(A) = w$, $W(B) = W(N) - w$.*

Proof. A set of trees whose total weight is the largest power of two not exceeding $W(N)$ can be achieved by successively adding the largest trees in N until the cumulative weight is that power of two; because larger pieces are added before smaller pieces, this power of two cannot be overshot. Allocate these trees to either A or B, according to the corresponding bit in the binary representation of w. Repeat this process until A and B reach the desired weight. ∎

DEFINITION 14. Consider the achievable values of $W(A)$. Let Quantum $(W(N))$ be the largest gap between achievable values.

LEMMA 15. *If* $2^{j-1} \leqslant W(N) < 2^j$ *then* Quantum$(W(N)) \leqslant 2^j - W(N)$.

Proof. Compute $i, 0 \leqslant i \leqslant j-1$, such that $2^j - 2^i \leqslant W(N) < 2^j - 2^{i-1}$. The binary representation of $W(N)$ has ones in positions corresponding to $2^i, 2^{i+1}, ..., 2^{j-1}$ (and zero in the position corresponding to 2^{i-1}). By Lemma 13, for each $h, 0 \leqslant h < 2^{j-i}$, the values $h2^i$ and $h2^i + (W(N) - (2^j - 2^i))$ are achievable. The gaps between these weights are $W(N) - (2^j - 2^i)$ and $2^j - W(N)$. By choice of i the latter gap is the larger of the two, and serves as a bound on Quantum$(W(N))$. ∎

LEMMA 16. *For any* $b_2 \leqslant b_1$, *if* $w \leqslant b_1, w + W(N) \geqslant b_2$, *and* Quantum $(W(N)) \leqslant b_1 - b_2$, *then there is a partition* $N = A \oplus B$ *such that* $b_2 \leqslant w + W(A) \leqslant b_1$.

Proof. If $b_2 - w \leqslant 0$, then $b_2 \leqslant w \leqslant b_1$, and setting $A = \varnothing$ implies the result. Suppose that $b_2 > w$. If $A = N$ then $w + W(A) \geqslant b_2$, so there is a choice of A of minimal weight such that $w + W(A) \geqslant b_2$. Since $b_2 - w > 0$ and Quantum$(W(N)) \leqslant b_1 - b_2$, $w + W(A) \leqslant b_2 + b_1 - b_2 = b_1$. ∎

Using the above lemmas we give a partition that minimizes $\max\{\mathscr{W}_A, \mathscr{W}_B\}$. Assume, without loss of generality, that $W(N_L) \geqslant W(N_R)$. We consider two cases.

Case 1. Suppose $W(N_{LR}) + W(N_L) \geqslant \frac{3}{4}D$. Then, since $W(N_{LR}) + W(N_L) < D$, and D is a power of two, we know that the binary representation of $\frac{3}{4}D$ is a subset of the representation of $W(N_L) + W(N_{LR})$. By Lemma 13, we can choose A_{LR} and A_L so that $W(A_{LR}) + W(A_L) = \frac{3}{4}D$. We set A_R and A_X to be the empty set, and thus $W(A_R) = W(A_X) = 0$. We get that $\mathscr{W}_A = \frac{3}{8}D$ and $\mathscr{W}_B < D - \frac{3}{4}D = \frac{1}{4}D$. Thus $\max\{\mathscr{W}_A, \mathscr{W}_B\} = \frac{3}{8}D$, a contradiction.

Case 2. Suppose $W(N_L) + W(N_{LR}) < \frac{3}{4}D$, so that $\frac{1}{2}W(N_L) + \frac{1}{2}W(N_{LR}) < \frac{3}{8}D$. To make the presentation clearer, we add "dummy" nodes to N_X to achieve the equality:

$$W(N_{LR}) + W(N_L) + W(N_R) + W(N_X) + 1 = D.$$

Clearly, this can be done without loss of generality. From our assumption that $W(N_R) \leqslant W(N_L)$, we get $W(N_R) < \frac{1}{2}D$, hence

$$\tfrac{1}{2}W(N_{LR}) + \tfrac{1}{2}W(N_L) + W(N_X) \geqslant \tfrac{1}{2}(D - W(N_R) - 1) \geqslant \tfrac{1}{4}D.$$

Furthermore, since $W(N_X) < \frac{1}{4}D$ (see Inequality (3)), by Lemma 15, Quantum($W(N_X)$) $\leqslant (1/8)\,D$. Thus, by Lemma 16, setting $w = \frac{1}{2}W(N_{LR}) + \frac{1}{2}W(N_L)$, $N = N_X$, $b_1 = \frac{3}{8}D$, and $b_2 = \frac{1}{4}D$, we can select A_X so that

$$\tfrac{1}{4}D \leqslant \tfrac{1}{2}W(N_{LR}) + \tfrac{1}{2}W(N_L) + W(A_X) \leqslant \tfrac{3}{8}D.$$

Setting $A_{LR} = N_{LR}$, $A_L = N_L$, $A_R = \varnothing$, and correspondingly $B_{LR} = B_L = \varnothing$, $B_R = N_R$, we get $\frac{1}{4}D \leqslant \mathscr{W}_A \leqslant \frac{3}{8}D$. Since $A_R = B_L = \varnothing$, we have

$$\begin{aligned}\mathscr{W}_A + \mathscr{W}_B &= (1/2)(W(N_{LR}) + W(N_L) + W(N_R) + 2W(N_X)) \\ &< \tfrac{1}{2}(D + W(N_X)) < \tfrac{5}{8}D.\end{aligned}$$

Since $\mathscr{W}_A \geqslant \frac{1}{4}D$, $\mathscr{W}_B < \frac{3}{8}D$. Thus, $\max\{\mathscr{W}_A, \mathscr{W}_B\} \leqslant \frac{3}{8}D$, a contradiction.

This establishes the induction when $k = 2j$ is even.

Now, consider the case when $k = 2j+1$ is odd. Set $U \equiv 2^{3j-1}$. By the assumption $D(2j+1) \leqslant 3 \cdot 2^{3j-1} = 3U$. On the other hand, by inductive hypothesis we have $D(2j) > 2^{3j-1} = U$. Again, the proof is by contradiction. We consider the same three different subtrees of T that compute some $\tilde{P}_{(k-1,0)}$ as in the even case and show that the weight of one of them is at most $U = 2^{3j-1}$, a contradiction.

From the first two subtrees of T we get the following inequalities which correspond to Inequalities (1), (2),

$$\tfrac{1}{2}W(N_{LR}) + W(N_R) \geqslant D(k-1) > U, \tag{4}$$

$$\tfrac{1}{2}W(N_{LR}) + W(N_L) \geqslant D(k-1) > U. \tag{5}$$

As in the even case the weight of the third tree is at most $\max\{\mathscr{W}_A, \mathscr{W}_B\}$; we show how to minimize this value. Assume, without loss of generality, that $W(N_L) \geqslant W(N_R)$. Again, we consider two cases.

Case 1. Suppose $2U \leqslant W(N_{LR}) + W(N_L) < 3U$. By Lemma 13, we can choose A_{LR} and A_L so that $W(A_{LR}) + W(A_L) = 2U$. We set $A_R = A_X = \varnothing$. Then, as before, $\mathscr{W}_A = U$, $\mathscr{W}_B \leqslant 3U - 2U = U$, and $\max\{\mathscr{W}_A, \mathscr{W}_B\} = U$, a contradiction.

Case 2. Suppose $W(N_{LR}) + W(N_L) < 2U$. Let $d_1 = 2U - (W(N_{LR}) + W(N_L))$. As before, add "dummy" nodes to N_X to achieve the equality:

$$W(N_{LR}) + W(N_L) + W(N_R) + W(N_X) + 1 = 3U.$$

By Inequalities (4), (5), we get $W(N_X) < U$. Let $d_2 = U - W(N_X)$, and thus $W(N_R) = d_1 + d_2 - 1$. We consider two subcases.

Case 2.1. Suppose $d_2 \geqslant d_1$. Then, set $A_{LR} = N_{LR}$, $A_L = N_L$, and $A_R = A_X = \varnothing$, correspondingly set $B_{LR} = B_L = \varnothing$, $B_R = N_R$, and $B_X = N_X$. We get

$$\mathscr{W}_A = \tfrac{1}{2} W(N_{LR}) + \tfrac{1}{2} W(N_L) < U,$$
$$\mathscr{W}_B = \tfrac{1}{2} W(N_R) + W(N_X) = U + \tfrac{1}{2}(d_1 - 1 - d_2) < U,$$

a contradiction.

Case 2.2. Suppose $d_2 < d_1$. By Lemma 15, Quantum$(W(N_X)) \leqslant d_2$. Observe that $\frac{1}{2} W(N_{LR}) + \frac{1}{2} W(N_L) < U + \frac{1}{2} d_2$, and

$$\begin{aligned}\tfrac{1}{2} W(N_{LR}) + \tfrac{1}{2} W(N_L) + W(N_X) &> \tfrac{1}{2} W(N_R) + W(N_X) \\ &= U + \tfrac{1}{2}(d_1 - 1 - d_2) \geqslant U > U - \tfrac{1}{2} d_2.\end{aligned}$$

Then, by Lemma 16, we can select A_X so that

$$U - \tfrac{1}{2} d_2 \leqslant \tfrac{1}{2} W(N_{LR}) + \tfrac{1}{2} W(N_L) + W(A_X) \leqslant U + \tfrac{1}{2} d_2.$$

Set $A_{LR} = N_{LR}$, $A_L = N_L$, and $A_R = \varnothing$, correspondingly $B_{LR} = B_L = \varnothing$, $B_R = N_R$. We get $U - \frac{1}{2} d_2 \leqslant \mathscr{W}_A \leqslant U + \frac{1}{2} d_2$, and

$$\begin{aligned}\mathscr{W}_A + \mathscr{W}_B &= \tfrac{1}{2}(W(N_{LR}) + W(N_L) + W(N_R) + 2W(N_X)) \\ &< (3/2)\ U + \tfrac{1}{2} W(N_X) = 2U - \tfrac{1}{2} d_2.\end{aligned}$$

This implies that $\mathscr{W}_B \leqslant U$. If $\mathscr{W}_A \leqslant U$, then we already obtain a contradiction. Suppose $U < \mathscr{W}_A \leqslant U + \frac{1}{2} d_2$. Let $\mathscr{A}$ be the set of trees in Left(A_{LR}), Left(A_L), and A_X, and let $\mathscr{B}$ be the set of trees in Right(B_R) and B_X. Since the binary representation of $W(\mathscr{A}) = \mathscr{W}_A$ contains the binary representation of U as a subset, then by Lemma 13 we can select a subset of the trees in $\mathscr{A}$ achieving the weight U exactly. Remove the remaining trees from $\mathscr{A}$ and add them to the set of trees in $\mathscr{B}$. Now, compute the new values of $\mathscr{W}_A$ and $\mathscr{W}_B$. Clearly, the new value of $\mathscr{W}_A$ is U. To bound the new value of $\mathscr{W}_B$, we have to estimate the effect of the trees added to $\mathscr{B}$. The effect of these trees will be at most doubled, since in the worst case a tree from A_L is moved to B_L. The total weight of the removed trees in $\mathscr{A}$ is at most $\frac{1}{2} d_2$, thus the new total value of $\mathscr{A}$ and $\mathscr{B}$ is at most $\frac{1}{2} d_2$ more than their old total value, which is $2U - \frac{1}{2} d_2$. Thus, the new total value is at most $2U$, and the new value of $\mathscr{W}_B$ is at most U; a contradiction. ▌

4. OPEN PROBLEMS

It seems that the correct lower bound for the binary leaf-addition expression should be $2 \log n - O(1)$. This would close the gap with Kosaraju's upper bound. Also, no non-trivial lower bounds for the Boolean monotone case are known.

ACKNOWLEDGMENTS

We thank Alok Aggarwal for introducing us to the problem and for many fruitful discussions. We are also grateful to Uri Feige and Allan Borodin for their help.

REFERENCES

[B74] R. Brent, The parallel evaluation of general arithmetic expressions, *J. Assoc. Comput. Mach.* **21** (1974), 201–206.

[BKM73] R. Brent, D. Kuck, and K. Maruyama, Parallel evaluation of arithmetic expressions without division, *IEEE Trans. Comput.* **22** (1973), 532–534.

[BM75] A. Borodin and I. Munro, "The Computational Complexity of Algebraic and Numeric Problems," Amer. Elsevier, New York, 1975.

[JS82] M. Jerrum and M. Snir, Some exact complexity results for straight-line computations over semirings, *J. Assoc. Comput. Mach.* **29** (1982), 874–897.

[Kos86] S. R. Kosaraju, Parallel evaluation of division free arithmetic expressions, *in* "Proc. of the 18th ACM Symp. on Theory of Computing, May 1986," pp. 231–239.

[KM75] D. Kuck and K. Maruyama, Time bounds on the parallel evaluation of arithmetic expressions, *SIAM J. Comput.* **4** (1975), 147–162.

[Mil75] W. Miller, Computer search for numerical instability, *J. Assoc. Comput. Mach.* **22** (1975), 512–521.

[MP76] D. E. Muller and F. P. Preparata, Restructuring of arithmetic expressions for parallel evaluation, *J. Assoc. Comput. Mach.* **23** (1976), 534–543.

[PM76] F. P. Preparata and D. E. Muller, Efficient parallel evaluation of Boolean expressions, *IEEE Trans. Comput.* **25** (1976), 548–549.

[PMB77] F. P. Preparata, D. E. Muller, and A. B. Barak, Reduction of depth of Boolean networks with a fan-in constraint, *IEEE Trans. Comput.* **26** (1977), 474–479.

[SS80] E. Shamir and M. Snir, On the depth complexity of formulas, *Math. Systems Theory* **13** (1980), 301–322.

[Sni91] M. Snir, Size depth trade-offs for monotone arithmetic circuits, *Theoret. Comput. Sci.* **22** (1991), 85–93.