

Exact complexity results for path polynomials over semirings

Andrew Lyons

Department of Computer Science, Dartmouth College

lyonsam@gmail.com

Abstract

For a directed acyclic graph (DAG) whose arcs are labeled with indeterminates, the *path polynomial* associated with a source-sink pair is the sum, over all paths from the source to the sink, of the product of the indeterminates along that path. Polynomials of this form arise in a wide variety of contexts. The path polynomials arising from layered DAGs are exactly the polynomials computing chained products of matrices with arbitrary dimensions and sparsity pattern. When evaluated over certain appropriate semirings, these polynomials capture many important graph problems such as shortest paths, minimum bottleneck paths, connectivity (reachability), and others. Path polynomials also arise in the context of algorithmic differentiation, where they represent the evaluation of derivatives according to the chain rule.

We consider the computation of path polynomials using monotone arithmetic circuits. This model is chosen both because of the feasibility of proving exact lower bounds and because it is a natural model of computation over semirings. We give exact lower bounds on the number of additions, multiplications, and total number of arithmetic operations needed for certain classes of DAGs. In each of these cases, we also give polynomial-time algorithms for constructing circuits that are provably optimal in all three of these measures. Some of our bounds are obtained by formulating a collection of reduction rules that may be applied to make the DAG (and thus the corresponding path polynomials) simpler. Aside from the general utility of these rules, we are able to show that certain classes of DAGs can be completely reduced, yielding optimal circuits.

1 Introduction

Let G be a directed acyclic graph (DAG). The *path polynomial* $p_{ij}(G)$ for source-sink pair (s_i, t_j) is defined as

$$p_{ij}(G) = \sum_{M \in [s_i \rightsquigarrow t_j]} \prod_{(u,v) \in M} x_{uv},$$

where $[s_i \rightsquigarrow t_j]$ denotes the set of paths from s_i to t_j in G and x_{uv} denotes an indeterminate associated with the arc (u, v) . This class of polynomials has many interesting properties and arises in many applications. We mention a few of them here.

We say a DAG is *k-homogeneous* if every path from a source to a sink consists of exactly k arcs.¹ If G is a k -homogeneous DAG, then $p_{ij}(G)$ computes entry P_{ij} of the matrix $P = X_1 X_2 \cdots X_k$, where the number of vertices on level ℓ corresponds to the number of rows in X_ℓ and the presence or absence of arcs from level ℓ to level $\ell + 1$ represents the sparsity pattern of X_ℓ . Moreover, any such matrix product can be represented in this way. This correspondence has also been observed by Vassilevska [14] and earlier by Mahr [9], Yuster and Zwick [15], and others.

¹ Such DAGs are often called *levelled* or *layered*. We choose to call them k -homogeneous to emphasize that the corresponding path polynomials are homogeneous.

Path polynomials play a large role in the theory of algorithmic differentiation [3]. Here, the vertices in the DAG represent variables in a block of computer code such that values at the sinks are computed as functions of the values at the sources. Each arc is then labeled with an indeterminate corresponding to the partial derivative of its target with respect to its source (usually some real value), and, as a consequence of the chain rule, the path polynomial $p_{ij}(G)$ computes exactly the derivative of t_j with respect to s_i . Thus the structure of G is known at compile time, but the indeterminates that label the arcs in G do not take values until runtime (and the program is often run many times with different input values). Here, $p_{ij}(G)$ computes entry J_{ij} of the Jacobian matrix of the function represented by G , and the problem of finding the best way to compute this matrix with a monotone arithmetic circuit is known as the OPTIMAL JACOBIAN ACCUMULATION PROBLEM.

Path polynomials correspond to many important graph problems when evaluated over different semirings. When evaluated over the semiring $\mathcal{M} = \langle \mathbb{R} \cup \{+\infty\}, \min, +, +\infty, 0 \rangle$ (as opposed to, say, the real semiring $\mathcal{R} = \langle \mathbb{R}, +, \times, 0, 1 \rangle$), the “sum” operation corresponds to \min and the “product” operation corresponds to the usual $+$ operation over the real numbers. The function computed by $p_{ij}(G)$ is then the length of the shortest path from source s_i to sink t_j . Other important problems are obtained by using other semirings; examples include minimum bottleneck paths, counting the number of paths, and reachability among source-sink pairs. We give a more detailed discussion of the computation of path polynomials over different semirings in Section 2.1.

1.1 Our approach

We are interested in the complexity of computing path polynomials in the model of arithmetic circuits. We shall be interested in not only the multiplicative complexity, but also the additive complexity and total complexity (both multiplications and additions). Our motivation for this is the observation that the generalized operations \oplus and \otimes may have different relative costs in different semirings. Our goal is to establish lower bounds on these measures and design efficient constructive algorithms to achieve them in the form of optimal arithmetic circuits.

Our approach encompasses all of the above-mentioned scenarios in the following sense: the context may be thought of as that of a compiler (or a more general pre-processing step) which constructs a circuit in the form of straight-line code. Here it is typical for the structure of the problem to be known beforehand (such as the structure (including sparsity) of a chained matrix computation), and we may assume that one or more path polynomial will be evaluated many times with different values for the indeterminates. An example is a DAG that will be queried for shortest source-sink paths as the arc weights change. This approach has particular relevance when the number of queries is much larger than the size of the graph, which may be modest. Indeed, as the number of queries increases, so does the importance of minimizing the size of the circuit.

In order to obtain tight bounds, we restrict our arithmetic circuits to be monotone. In doing so, we place our focus on exploiting the *structure* of the polynomial to be computed (derived from the DAG). Monotone arithmetic circuits make use of only the simplest ring axioms (associativity, distributivity, etc.). Circuits of this type are *universal* in a sense—they can be re-fashioned for use over any (semi)ring. This is to be considered an alternative to algorithms such as Strassen’s method for multiplying two matrices [13], which assumes that the matrices are dense and exploits the more complex algebraic properties of the underlying field. We wish to avoid the high constant overhead that is typically associated with algorithms of this type. Furthermore, approaches of this type can only reasonably be expected for certain fixed problems, and may not exhibit satisfactory numerical stability in all cases. Nonetheless, if these issues are not of concern, we note that Strassen’s algorithm beats the known lower bounds for monotone matrix multiplication, and does

so (if we are to count only multiplications) even in the case of multiplying two dense 2×2 matrices.

1.2 Previous work

Jerrum and Snir [7] gave a number of exact complexity results for computations over semirings. Of particular relevance is their result that $(k-1)n^3$ multiplications are necessary and sufficient to evaluate the product $X_1X_2\cdots X_k$ of k dense $n \times n$ matrices over the Boolean semiring $\mathcal{B} = \langle \{0, 1\}, \vee, \wedge, 0, 1 \rangle$. The lower bound is met from above by simply evaluating the product from left to right in the naive way. This result is shown to extend to the semirings \mathcal{R} and \mathcal{M} by the description of a homomorphism that takes any computation of this function over \mathcal{R} or \mathcal{M} to a computation over \mathcal{B} . We will use a variant of this technique later. As mentioned at the beginning of this section, this result is immediately a result concerning the complexity of evaluating the collection of path polynomials arising from a k -homogeneous DAG G in which all layers have the same number of vertices and all possible edges from one layer to the next are present in G . Prior to our work, nothing was known about the complexity of (collections of) path polynomials with less rigid structure.

There has also been a good deal of work on methods for evaluating chained products of matrices of different dimensions; this problem has been subject to a formulation as finding an optimal triangulation of a polygon [4, 5] and can be solved in polynomial time (a canonical example for dynamic programming). These methods, however, are designed for dense matrices specifically, and restrict the computation of the product to some bracketing scheme among the matrices. This amounts to exploiting only associativity and only at the granularity of entire matrices. What if the matrices have some known sparsity pattern?

While no other lower bounds were previously known, we note an upper bound (though it is not tight) that follows from algorithms used widely in practice. In particular, while the number of monomials in a path polynomial p_{ij} could be exponential in the size of G , p_{ij} can nonetheless be evaluated using no more than $|A(G)|$ multiplications, where $A(G)$ is the set of arcs of G . This can be accomplished by traversing the vertices v reachable from s_i in topological order, essentially computing $p_{iv}(G)$ (to abuse notation a bit) for each one. This is sometimes described as a form of dynamic programming, as $p_{iv}(G)$ can be computed easily from the polynomials corresponding to the predecessors of v in the topological order.

In algorithmic differentiation (AD), this is known as the *forward mode* of derivative accumulation (the *reverse mode*, which entails the traversal of G in reverse topological order, gives the same bound). It was noted by Iri [6] that these algorithms have natural interpretations over $\langle \mathbb{R} \cup \{+\infty\}, \min, +, +\infty, 0 \rangle$ and in fact correspond to well-known algorithms for shortest path computations. This connection with shortest paths is also discussed by Mahr [8, 9], and is considered by Mehlhorn [10] for general directed graphs (those not necessarily acyclic).

In the context of AD, Griewank and Naumann [2] gave a heuristic that is based on an adaption of the matrix chain bracketing technique described above. In their variant, any DAG can be expressed in terms of a chained product of extremely sparse matrices. Some attempt is made to prune rows and columns of the matrices that are entirely zero, though no concrete bounds are stated.

1.3 Results

We give a number of results concerning the complexity (in terms of monotone arithmetic circuits) of computing a single path polynomial. We can consider such polynomials to arise from DAGs with a single source s and a single sink t ; we will refer to such DAGs as *st-DAGs* and refer to the corresponding path polynomial as $p(G)$ (omitting the subscript ij).

The arithmetic circuits model of computation assumes that arbitrary constants from the underlying field are available for use in the computation of a polynomial. In Section 3, we show that no optimal monotone circuit for the path polynomial of an *st*-DAG can make use of such constants.

In Section 4, we derive exact lower bounds for path polynomials of series-parallel *st*-DAGs and complete *st*-DAGs. These bounds are obtained by showing that DAGs falling into one of these two categories can be reduced to a single edge by repeated application of certain *reduction rules*. The results for series-parallel graphs exhibit a difference of a factor approaching two between optimal circuits and those obtained using the standard forward or reverse modes. For complete *st*-DAGs, we show that the forward and reverse modes are both optimal, meeting the lower bound of $|A(G)| - |V(G)| + 1$ multiplications and the same number of additions.

In Section 5, we obtain exact lower bounds for computing the path polynomial of 3-homogeneous *st*-DAGs. Our result expresses the multiplicative complexity in terms of the size of an optimal vertex cover of the bipartite graph induced by the middle layer of arcs, and we show that the problem of constructing an optimal circuits for these DAGs is exactly as hard as the problem of constructing optimal vertex covers for bipartite graphs. In terms of chained matrix products, $p(G)$ in this case corresponds to the multiplication of a sparse matrix by vectors on either side. It follows from our results that there are cases for which the computation performed by an optimal monotone circuit cannot be expressed as either of the two possible bracketings, thus providing justification for a more fine-grained approach to general chained sparse matrix products.

All our results are demonstrated for computations of path polynomials over the real semiring \mathcal{R} . However, we are able to leverage a result by Jerrum and Snir [7] to show that the exact bounds we obtain (over \mathcal{R}) are also exact bounds over \mathcal{M} , and thus apply equally to shortest path computations.

2 Preliminaries

In this section, we define the computational model that we will use in proving our bounds. A formal description of arithmetic circuits is deferred to the latter half of the section, where we also discuss the particular properties of monotone arithmetic circuits.

2.1 Computing polynomial functions over semirings

A *semiring* \mathcal{S} is a system $\langle \mathbb{F}, \oplus, \otimes, 0_{\mathcal{S}}, 1_{\mathcal{S}} \rangle$ where \mathbb{F} is a set and the following properties are satisfied: (i) both \otimes and \oplus are commutative; (ii) both \otimes and \oplus are associative; (iii) \otimes distributes over \oplus ; (iv) $0_{\mathcal{S}}$ is the additive identity/multiplicative annihilator; and (v) $1_{\mathcal{S}}$ is the multiplicative identity. Note in particular that additive inverses are assumed not to exist, which means that cancellations do not occur in computations over semirings.

The results presented in this paper apply mainly to the following semirings.

The *real semiring* \mathcal{R} is defined by $\langle \mathbb{R}, +, \times, 0, 1 \rangle$, where $+$ and \times are just the usual addition and multiplication for real numbers.

The *min-plus semiring* \mathcal{M} is defined by $\langle \mathbb{R} \cup \{+\infty\}, \min, +, +\infty, 0 \rangle$. Note that \min corresponds to the generalized operation \oplus (and $0_{\mathcal{M}} = +\infty$), whereas $+$ in this case corresponds to the generalized operation \otimes (and $1_{\mathcal{M}} = 0$). As mentioned in Section 1, this semiring is suitable for computing shortest path functions.

We now find it necessary to distinguish between the computation of a formal polynomial p over a semiring \mathcal{S} and the evaluation of a *polynomial function* over \mathcal{S} . To illustrate this, we point out a

key difference between \mathcal{R} and \mathcal{M} , namely that \oplus is *idempotent* in \mathcal{M} (meaning that $x \oplus x = x$ for all $x \in \mathbb{F}$) but not in \mathcal{R} . This means that the formal polynomials represented by x and $x \oplus x$ both compute the same polynomial function x . For a polynomial function p , we denote by $\mathbf{C}_{\times}^{\mathcal{S}}(p)$ ($\mathbf{C}_{+}^{\mathcal{S}}(p)$) the minimum number of \otimes (\oplus) operations required to evaluate p over \mathcal{S} . Similarly, $\mathbf{C}^{\mathcal{S}}(p)$ denotes the total number of operations (both \otimes and \oplus) required. In general, there may be many formal polynomials over \mathcal{S} that compute the same polynomial function. For example, the formal polynomials over \mathcal{M} represented by x and $x \oplus x$ compute the same polynomial function x . It is important to note that the above complexity measures for polynomial functions are meant to refer to the minimal complexity over all formal polynomials computing the function. We refer the interested reader to the work of Jerrum and Snir [7], for a more in-depth treatment of these subtleties. While the flexibility afforded by idempotence may be viewed as an extra power inherent in \mathcal{M} , they were able to identify some polynomial functions for which this type of idempotence does not help.

Theorem 1 ([7]). *If p is a multilinear polynomial, then $\mathbf{C}_{\times}^{\mathcal{R}}(p) = \mathbf{C}_{\times}^{\mathcal{M}}(p)$ and $\mathbf{C}_{+}^{\mathcal{R}}(p) = \mathbf{C}_{+}^{\mathcal{M}}(p)$.*

Since path polynomials are always multilinear, this result implies that all of our bounds apply equally to computations over \mathcal{R} and \mathcal{M} . For convenience, we will work (almost) exclusively with the former.

2.2 Arithmetic circuits

Let \mathbb{F} be a field and X be a set of indeterminates. An *arithmetic circuit* Φ over X and \mathbb{F} is a directed acyclic graph² whose nodes are either *inputs* with zero children or *gates* with exactly two children. The inputs take labels from $X \cup \mathbb{F}$. Every gate is labeled with an operation in $\{\times, +\}$ and computes a polynomial in the ring $\mathbb{F}[X]$ in the natural way: gates labeled \times compute the product of their children; gates labeled $+$ compute the sum of their children. For the bulk of this paper we will consider only circuits that compute a single polynomial, and hence have exactly one node that is the *output*, which we will often denote by ϕ . The *size* of Φ , denoted $|\Phi|$, is the number of gates; $|\Phi|_{\times}$ and $|\Phi|_{+}$ denote the numbers of product and sum gates, respectively. For a node $\alpha \in \Phi$, $\text{inputs}(\alpha)$ denotes the set of inputs in Φ that have α as an ancestor, and $\text{mon}(\alpha)$ denotes the set of monomials of the polynomial computed at α . Since our operations are commutative, $\text{mon}(\alpha)$ itself will also be a set, the elements of which are the indeterminates that occur in the monomial.

2.3 Monotone circuits

A polynomial is *monotone* if the coefficient of every monomial is non-negative. An arithmetic circuit is *monotone* if every field element that occurs as a label on an input is positive. Whereas subtraction can be carried out in (general) arithmetic circuits by adding a value that has been multiplied with the constant -1 , monotone circuits can compute only monotone functions and are exactly those arithmetic circuits that do not make use of additive inverses (negative constants) from the underlying field. Monotone circuits therefore explicitly disallow subtraction, and thus cancellation as well.

We now state some basic facts about the structure of monotone circuits. Each of the following observations captures the intuition that nothing is destroyed in monotone computations.

Observation 2.1. *If α and γ are nodes in a monotone arithmetic circuit such that γ is an ancestor of α , then for every $A \in \text{mon}(\alpha)$ there exists some $Z \in \text{mon}(\gamma)$ such that $A \subseteq Z$.*

² To avoid confusion, we use tree terminology (node, parent, child, ancestor, etc.) for circuits, reserving digraph terms (vertex, arc, inarc, etc.) for G itself.

A polynomial is *homogeneous* if every monomial has the same degree. An arithmetic circuit is *homogeneous* if every node computes a homogeneous polynomial.

Observation 2.2. *Every monotone arithmetic circuit for a homogeneous polynomial is homogeneous.*

2.4 Multilinear circuits and parse trees

A polynomial is *multilinear* if it is linear in each input from X . A circuit is *multilinear*³ if every gate computes a multilinear polynomial.

Observation 2.3. *Every monotone circuit for a multilinear polynomial is multilinear.*

Observation 2.4 ([11]). *Let Φ be a monotone circuit for a multilinear polynomial. If ρ is a \otimes -gate in Φ with children $\{\alpha, \beta\}$, then $\text{inputs}(\alpha) \cap \text{inputs}(\beta) = \emptyset$.*

In this case, we say Φ is *multiplicatively disjoint*. Indeed, any higher-degree monomial computed by a monotone circuit cannot be subsequently removed, yet such monomials cannot occur in multilinear polynomials.

The structure of monotone multilinear circuits is captured nicely in the following way. We begin by defining *parse trees*, which chart the production of particular monomials in $\text{mon}(\Phi)$.

Definition 1 (parse tree [7, 1]). *A subcircuit T of Φ is a parse tree of Φ if it satisfies the following conditions:*

- (i) T contains the (unique) output of Φ .
- (ii) If T contains a sum gate σ , then T contains exactly one of the children of σ .
- (iii) If T contains a product gate ρ , then T contains both of the children of ρ .
- (iv) No proper subtree of T satisfies (i)-(iii).

We denote by $\text{PT}(\Phi)$ the set of all parse trees of Φ . For a parse tree $T \in \text{PT}(\Phi)$, let $\text{mon}(T)$ denote the monomial consisting of the inputs of T . From the definition given above, we have $\phi = \sum_{T \in \text{PT}(\Phi)} \text{mon}(T)$, where ϕ is the formal polynomial computed at the output of Φ .

3 Monic multilinear circuits and constant inputs

A polynomial is *monic* if the coefficient for every monomial is either 0 or 1. A circuit is monic if every gate computes a monic polynomial. Monotone monic circuits over \mathcal{R} can be thought of as *additively disjoint* in the following sense.

Observation 3.1. *Let Φ be a monotone monic circuit over \mathcal{R} . If σ is a \oplus -gate in Φ with children $\{\alpha, \beta\}$, then $\text{mon}(\alpha) \cap \text{mon}(\beta) = \emptyset$.*

Intuitively, this tells us that every monomial is computed exactly once for every gate, including the output gate(s). This property will be useful later on.

The goal of this section is to prove the following result.

³ In some cases, a distinction is drawn between multilinearity and *syntactic multilinearity* [12]. We note that the distinction is not necessary when considering monotone circuits, as every monotone multilinear circuit meets the conditions of being syntactically multilinear. Our definition corresponds to that of the latter.

Theorem 2. Let p be a monic multilinear polynomial that is either homogeneous or the path polynomial of some st -DAG. If Φ is a circuit for p that is optimal with respect to $|\Phi|$, $|\Phi|_{\times}$, or $|\Phi|_{+}$, then Φ is monic and has no constant inputs.

Recall that monotone circuits for homogeneous polynomials must be homogeneous, and that monotone circuits for multilinear polynomials must be multilinear (and are multiplicatively disjoint). It is not the case, however, that monotone circuits for monic polynomials must be monic. In particular, the algebraic properties of \mathcal{R} may lead to computations involving the use of constants as coefficients. Consider the circuit Φ computing $p(G)$ over \mathcal{R} shown in Figure 1.

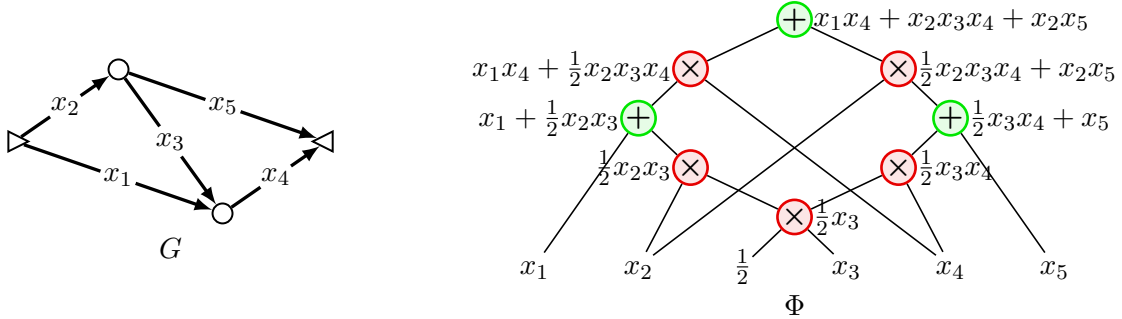


Figure 1: A st -DAG G and an arithmetic circuit Φ computing $p(G)$ over \mathcal{R} . Despite the fact that p is monic, Φ makes use of constants from the underlying field as inputs. We demonstrate that no such circuit can be optimal. The property of \mathbb{R} exploited by Φ is that, $\frac{1}{2} \oplus \frac{1}{2} = \frac{1}{2} + \frac{1}{2} = 1 = 1_{\mathcal{R}}$. Observe, however, that the circuit obtained by replacing the constant $\frac{1}{2}$ with the constant $1_{\mathcal{M}}$ computes $\min \{x_1x_4, x_2x_3x_4, x_2x_3x_4, x_2x_5\} = \min \{x_1x_4, x_2x_3x_4, x_2x_5\}$ over \mathcal{M} .

Note this is not the only way in which constants may be of use in a monotone monic circuit for a multilinear polynomial. Consider the monic polynomial $p = y + z + xy + xz$, which may well be computed as $(1_{\mathcal{S}} \oplus x) \otimes (y \oplus z)$ by a circuit over the semiring $\mathcal{S} \in \{\mathcal{R}, \mathcal{M}\}$ using only two \oplus -gates and a single \otimes -gate. We now demonstrate that constants cannot be used in this way if p is homogeneous or p is the path polynomial of some st -DAG G . For technical reasons, we constrain all constant inputs c in Φ to have exactly one parent in Φ . Clearly, if c has more than one parent, then we can simply create duplicate constant inputs (one for each parent, each with the same value as c) without affecting the number of gates in ϕ . Also, since it is pointless to include gates that compute constants, we will assume that every gate has at most one constant child.

Observation 3.2. If p is homogeneous and c is a constant input of Φ , then the parent of c in Φ is a \otimes -gate.

This follows from the fact that a sum gate with a constant child will only be homogeneous if it computes a constant. We now show an analogous result for path polynomials.

Lemma 3.3. If p is the path polynomial of some st -DAG G and c is a constant input of Φ , then the parent of c in Φ is a \otimes -gate.

Proof. Suppose the sole parent σ of c is a \oplus -gate and let α be the other child of σ (thus α computes a non-constant polynomial). Now let $T_c \in \text{PT}(\Phi)$ be any parse tree containing c . Since σ is a \oplus -gate, there will be at least one corresponding parse tree $T_\alpha \in \text{PT}(\Phi)$ that includes α instead of c , such that T_c and T_α are the same everywhere except below σ . It follows that there are distinct monomials $\text{mon}(T_c), \text{mon}(T_\alpha) \in \text{mon}(\phi)$. This is a contradiction, however, because

the indeterminates occurring in $\text{mon}(T_c)$ are a proper subset of those occurring in $\text{mon}(T_\alpha)$, which means that $\text{mon}(T_c)$ and $\text{mon}(T_\alpha)$ cannot both correspond to source-sink paths in G . \square

Lemma 3.4. *Let p be a monic multilinear polynomial that is either homogeneous or the path polynomial of some st -DAG. If Φ is a monotone circuit that computes p over \mathcal{R} , then the monotone circuit $\Phi_{\mathcal{M}}$ obtained by setting every constant input to $1_{\mathcal{R}} = 1$ computes p over \mathcal{M} .*

Proof. We may assume that every constant input c to Φ is nonzero (hence $\Phi_{\mathcal{M}}$ will have no input equal to $+\infty$). Suppose Φ has a constant input c . As we did above, we assume without loss of generality that c has precisely one parent in Φ which we will denote by ρ . It follows from Observation 3.2 and Lemma 3.3 that ρ must be a \otimes -gate with children $\{c, \alpha\}$ for some node α that computes a non-constant polynomial. It follows that ρ computes α when c is set to $1_{\mathcal{R}}$, and ρ may be contracted into α . Now let β be an arbitrary node in $\Phi_{\mathcal{M}}$ and observe that the set $\text{inputs}(\beta)$ is just the same as in ϕ . It follows by an inductive argument (starting at the inputs of $\Phi_{\mathcal{M}}$) that if β computes $b_1M_1 + b_2M_2 + \dots + b_kM_k$ in Φ for coefficients $b_1, \dots, b_k \in \mathbb{R}$, then β computes $M_1 + M_2 + \dots + M_k$ in $\Phi_{\mathcal{M}}$. Since the inductive argument extends all the way to the output of the circuit, and p is a monic polynomial, we have that $\Phi_{\mathcal{M}}$ computes p as desired. \square

Proof of Theorem 2. Let Φ be circuit for p that is optimal in terms of $|\Phi|, |\Phi|_{\times}$, or $|\Phi|_{+}$, and let $\Phi_{\mathcal{M}}$ be obtained as in the proof of Lemma 3.4. If Φ had a constant input, then we would have $|\Phi_{\mathcal{M}}| < |\Phi|$ (at least one \otimes -gate would be removed) which contradicts Theorem 1; hence Φ has no constant inputs. To see that Φ must also be monic, observe that any coefficient greater than one (as could be created by summing two identical monomials) cannot be destroyed without cancellations or multiplication with a monomial with a coefficient less than one; the former do not occur in monotone circuits, and the latter cannot be created in monotone circuits without constant inputs. It follows that Φ must be monic, which completes the proof. \square

4 Exact lower bounds and polynomial-time algorithms via reduction rules

Throughout this section, G will denote an arbitrary st -DAG. We consider transformations $G \rightarrow G'$ where we can give a precise relation between the complexity of $p(G)$ and that of $p(G')$. We shall refer to these transformations as *reduction rules*, as there is always some sense in which G' is simpler than G . Importantly, these rules also suggest that it is straightforward to construct a circuit that computes $p(G)$ from a circuit that computes $p(G')$.

In some cases, a sequence $G \rightarrow G' \rightarrow \dots \rightarrow G^{(k)}$ with $k = O(|A(G)|)$ reduces the graph to a single edge. We will call such graphs *completely reducible*. Since all of our reduction rules can be applied in polynomial time and at each step we have a direct relation of the complexity of $p(G^{(i)})$ and $p(G^{(i+1)})$, the result will be a polynomial-time algorithm that produces an optimal circuit for $p(G)$. At the end of this section, we show that both series-parallel and complete st -DAGs (see definitions below) are completely reducible with respect to the reduction rules we present here.

We also note that these rules can be applied to any graph, regardless of whether it is completely reducible. For this reason, these rules should be applied whenever possible (think of their application as a preprocessing step) yielding a simpler graph which may be easier to tackle via subsequent application of heuristics.

4.1 The reduction rules

Rule 1 (parallel reduction). *Let $u, v \in V(G)$ be any two vertices in G . If there exist distinct (parallel) arcs $a, b \in A(G)$ from u to v , then G' is obtained from G by removing b .*

Lemma 4.1. *The following hold whenever G' is obtained from G by an application of Rule 1.*

- (i) $\mathbf{C}_\times(p(G)) = \mathbf{C}_\times(p(G'))$;
- (ii) $\mathbf{C}_+(p(G)) = \mathbf{C}_+(p(G')) + 1$;
- (iii) $\mathbf{C}(p(G)) = \mathbf{C}(p(G')) + 1$.

Proof. Let x_a and x_b denote the indeterminates associated with arcs a and b , respectively.

(\leq): let Φ' be an arbitrary monotone circuit for $p(G')$. Replacing the input x_a in Φ' with a \oplus -gate with children x_a, x_b (thus computing $x_a + x_b$) will yield a circuit Φ that computes $p(G)$ where Φ has the same number of \otimes -gates and exactly one more \oplus -gate than Φ' .

(\geq): Now let Φ be an arbitrary monotone circuit for $p(G)$, and consider the circuit Φ' obtained from Φ by setting x_b to 0. This will set exactly the parse trees containing x_{vw} to 0, implying that Φ' computes $p(G')$. We now argue that simplifying Φ' will entail the removal at least one \oplus -gate, yielding $|\Phi'|_+ < |\Phi|_+$. Any \otimes -gate ρ with x_b as a child will now compute the zero polynomial, as will any \otimes -gate that is a parent of ρ , and so on. Let this process continue upwards in the circuit until each such trace of ancestry of x_b encounters a \oplus -gate; all such \oplus -gates have one child that computes the zero polynomial, and may thus be contracted to their other child. Finally, we note that we are guaranteed to encounter at least one such \oplus -gate, as not every path through G contains the arc labeled by x_b . It follows that $|\Phi'|_+ < |\Phi|_+$ as desired. \square

As the parallel reduction rule can be applied at any time without penalty, we will assume for the remainder of this paper that G does not contain any parallel arcs.

Rule 2 (series reduction). *If there exists some $v \in V(G)$ such that the only inarc to v is (u, v) and the only outarc of v is (v, w) , then G' is obtained from G by removing v along with both of its incident arcs and creating a new arc from u to w .*

This rule can also be thought of as a contraction of arc (v, w) . We also note that it is possible for an application of Rule 2 to result in a DAG with parallel arcs; we assume such edges will be subsequently removed using the parallel reduction rule.

Lemma 4.2. *If $(u, v) \in A(G)$ and there is no alternative path from u to v in G , then every parent of the input x_{uv} of Φ is a \otimes -gate.*

Proof. Suppose σ is a \oplus -gate and a parent of x_{uv} in Φ . Let $T \in \text{PT}(\Phi)$ be any parse tree containing x_{uv} . We have that T represents some monomial $M = M_1 x_{uv} M_2$ in $p(G)$. Now observe that there exists a parse tree T' that includes σ instead of x_{uv} , thus representing the monomial $M_1 M_\sigma M_2$ where $M_\sigma \in \text{mon}(\sigma)$. Since there is no alternative path from u to v , $M_1 M_\sigma M_2$ cannot represent a path in G , which is a contradiction. \square

Lemma 4.3. *The following hold whenever G' is obtained from G by an application of Rule 2.*

- (i) $\mathbf{C}_\times(p(G)) = \mathbf{C}_\times(p(G'))$;
- (ii) $\mathbf{C}_+(p(G)) = \mathbf{C}_+(p(G')) + 1$;
- (iii) $\mathbf{C}(p(G)) = \mathbf{C}(p(G')) + 1$.

Proof. (\leq): Let Φ' be an arbitrary monotone circuit for $p(G')$, and observe that we can obtain a circuit Φ that computes $p(G)$ by replacing the input x_{uw} in Φ' with a \otimes -gate ρ such that the children of ρ are x_{uv}, x_{vw} (ρ thus computes $x_{uv} \otimes x_{vw}$).

(\geq): Now let Φ be an arbitrary monotone circuit for $p(G)$ and consider the circuit Φ' obtained from Φ by setting x_{vw} to 1. By Lemma 4.2, every parent of x_{vw} in Φ is a \otimes -gate, and simplification of Φ' will mean contracting each parent of x_{vw} into its other child, which cannot be an ancestor of x_{vw} . It follows that $|\Phi|_{\times} < |\Phi'|_{\times}$. Since the parse trees affected by this process are exactly those containing x_{vw} , it follows that Φ' computes $p(G')$. \square

Rule 3 (absorption reduction). *If there exist distinct $u, v, w \in V(G)$ such that*

- (i) $(u, v) \in A(G)$ is the only inarc of v in G ,
- (ii) $(u, w), (v, w) \in A(G)$,
- (iii) there is no alternative path in G from v to w ,

then G' is obtained from G by removing arc (v, w) .

Lemma 4.4. *The following hold whenever G' is obtained from G by an application of Rule 3.*

- (i) $\mathbf{C}_{\times}(p(G)) = \mathbf{C}_{\times}(p(G')) + 1$;
- (ii) $\mathbf{C}_{+}(p(G)) = \mathbf{C}_{+}(p(G')) + 1$;
- (iii) $\mathbf{C}(p(G)) = \mathbf{C}(p(G')) + 2$.

Proof. (\leq): Let Φ' be an arbitrary monotone circuit computing $p(G')$. Consider the circuit obtained by replacing the input x_{uw} in Φ' with a \oplus -gate ρ with children $\{x_{uv}, \sigma\}$, where σ is a new \otimes -gate with children $\{x_{uv}, x_{vw}\}$ (thus ρ computes $x_{uv} + x_{uv}x_{vw}$). The result is a circuit Φ that computes $p(G)$ where $|\Phi|_{+} = |\Phi'|_{+} + 1$, $|\Phi|_{\times} = |\Phi'|_{\times} + 1$, and thus $|\Phi| = |\Phi'| + 1$ as desired.

(\geq): Now let Φ be an arbitrary monotone circuit computing $p(G)$ and consider the circuit Φ' obtained by setting x_{vw} to 0 in Φ . This will set exactly the parse trees containing x_{vw} to 0, implying that Φ' computes $p(G')$. We now argue that simplifying Φ' will entail removing at least one \otimes -gate and at least one \oplus -gate, yielding $|\Phi'|_{\times} < |\Phi|_{\times}$ and $|\Phi'|_{+} < |\Phi|_{+}$. First, observe that (by Lemma 4.2) every parent of x_{vw} in Φ is a \otimes -gate, and thus these gates now compute the zero polynomial. Any \otimes -gate parents of these gates will likewise compute the zero polynomial, and this process will continue upwards in the circuit until a \oplus -gate is encountered. When we encounter a \oplus -gate with a child that computes the zero polynomial, we may contract it to its other child, thus leaving the other parse trees unaffected. Because $(v, w) \in A(G)$ is not contained in every s - t path in G , this process must encounter at least one \oplus -gate along the way, which completes the proof. \square

4.2 Optimal circuits in polynomial time for some completely reducible classes of st -DAGs

The class of *series-parallel st -DAGs* is defined inductively as follows.

- (i) An st -DAG consisting of a single arc is a series-parallel st -DAG.
- (ii) If G_1, G_2 are series-parallel st -DAGs, then so is the st -DAG obtained by identifying the sink of G_1 with the source of G_2 (series composition).
- (iii) If G_1, G_2 are series-parallel st -DAGs, then so is the st -DAG obtained by identifying the source of G_1 with the source of G_2 and identifying the sink of G_1 with the sink of G_2 (parallel composition).
- (iv) There are no further series-parallel st -DAGs.

Theorem 3. *There exists a polynomial-time algorithm that, given a series-parallel st -DAG G , produces a monotone circuit Φ for $p(G)$ satisfying $|\Phi|_{\times} = \mathbf{C}_{\times}(p(G))$, $|\Phi|_{+} = \mathbf{C}_{+}(p(G))$, and $|\Phi| = \mathbf{C}(p(G))$.*

Proof. Let G be a series-parallel st -DAG. By the definition given above, any such st -DAG G can be constructed by a sequence of series and parallel compositions. It follows that G can be reduced to a single edge by reversing the sequence and applying corresponding series (Rule 2) and parallel (Rule 1) reductions. \square

An st -DAG G is a *complete st -DAG* if for every distinct $u, v \in V(G)$, either $(u, v) \in A(G)$ or $(v, u) \in A(G)$. Note that the vertices of a complete st -DAG can be assigned a natural linear ordering by their indegree.

Theorem 4. *There exists a polynomial-time algorithm that, given a complete st -DAG G , produces a monotone circuit Φ for $p(G)$ satisfying $|\Phi|_{\times} = \mathbf{C}_{\times}(p(G))$, $|\Phi|_{+} = \mathbf{C}_{+}(p(G))$, and $|\Phi| = \mathbf{C}(p(G))$.*

Proof. Let G be a complete st -DAG. Our algorithm can be thought of as consisting of two distinct phases. In the first phase, we apply absorption reductions (Rule 3) to all arcs that are incident on neither the source nor the sink. Let (s, v_1, \dots, v_n, t) be the unique topological ordering of G . We have that v_1 has a unique inarc, and for every arc $(v, v_i) \in A(G)$ there is a corresponding arc $(s, v_i) \in A(G)$. It follows that all such (v, v_i) can be removed by application of Rule 3. As a result, v_2 is left with only one inarc, and the same process can be applied to this vertex. Continuing in this fashion, every arc that is not incident on the source or sink can be removed, resulting in a series-parallel st -DAG. The second phase reduces this resulting st -DAG to a single edge as in the proof of Theorem 3, which completes the proof. \square

5 3-homogeneous st -DAGs

Here and throughout this section, G will be an arbitrary 3-homogeneous st -DAG with source s and sink t . Let U and W the first and second layers of vertices in G , respectively, so that $V(G) = \{s\} \cup U \cup W \cup \{t\}$. Let $X_{uw} \subset X$ denote the set of indeterminates associated with the arcs (u_i, w_j) that constitute the middle layer of G . Note that every monomial in $p(G)$ contains exactly one indeterminate from X_{uw} ; this fact will play a large role in our proofs.

A *vertex cover* for an undirected graph \hat{G} is a set $C \subseteq V(\hat{G})$ of vertices such that every edge in $E(\hat{G})$ has at least one endpoint in C . The *vertex cover number* $\tau(\hat{G})$ of \hat{G} is the smallest integer k such that there exists a vertex cover of size k for \hat{G} . Let $G[U \cup W]$ denote the undirected bipartite graph induced by the middle layer of our DAG G (the directed arcs become undirected *edges*). Our results relate the complexity of $p(G)$ to $\tau(G[U \cup W])$. Moreover, we will show that the existing polynomial-time algorithms for constructing an optimal vertex cover of a bipartite graph can be used to construct an optimal circuit for $p(G)$.

We can now state the main result of this section.

Theorem 5. *If G is a 3-homogeneous st -DAG, then*

- (i) $\mathbf{C}_{\times}(p(G)) = |X_{uw}| + \tau(G[U \cup W])$;
- (ii) $\mathbf{C}_{+}(p(G)) = |X_{uw}| - 1$;
- (iii) $\mathbf{C}(p(G)) = 2|X_{uw}| + \tau(G[U \cup W]) - 1$.

Moreover, there exists a polynomial-time algorithm that constructs a monotone circuit for $p(G)$ that meets all of these bounds simultaneously.

5.1 Upper bounds

We begin with the upper bound in order to give some intuition for the structure of optimal circuits for these graphs.

Lemma 5.1. *If G is a 3-homogeneous st -DAG, then*

- (i) $\mathbf{C}_\times(p(G)) \leq |X_{uw}| + \tau(G[U \cup W])$;
- (ii) $\mathbf{C}_+(p(G)) \leq |X_{uw}| - 1$;
- (iii) $\mathbf{C}(p(G)) \leq 2|X_{uw}| + \tau(G[U \cup W]) - 1$.

Proof. Let $H \subseteq U \cup W$ be a minimum vertex cover for $G[U \cup W]$. We will create a collection $G_1, G_2, \dots, G_{|H|}$ of 3-homogeneous st -DAGs that represent a partition of X_{uw} , so that $p(G) = \sum_{k=1}^{|H|} p(G_k)$. Our partition places (u_i, w_j) in the graph G_k where k is the smallest integer satisfying $u_i = h_k$ or $w_j = h_k$. Accordingly, we place $(s, u_i), (w_j, t) \in A(G_k)$ whenever $(u_i, w_j) \in A(G_k)$. Since H is a vertex cover, it follows that every (u_i, w_j) will be included in exactly one G_k . This is a natural partition of the monomials: if $h_k \in U$, then every monomial in $p(G_k)$ includes the indeterminate x_{sh_k} ; otherwise, $h_k \in W$ and every monomial in $p(G_k)$ includes the indeterminate $x_{h_k t}$. Define

$$I_k = \begin{cases} \{j : (h_k, w_j) \in A(G_k)\} & \text{if } h_k \in U, \\ \{i : (u_i, h_k) \in A(G_k)\} & \text{if } h_k \in W, \end{cases}$$

from which we obtain

$$p(G_k) = \begin{cases} x_{sh_k} \left(\sum_{j \in I_k} x_{h_k w_j} x_{w_j t} \right) & \text{if } h_k \in U, \\ \left(\sum_{i \in I_k} x_{s u_i} x_{u_i h_k} \right) x_{h_k t} & \text{if } h_k \in W. \end{cases}$$

It follows that we can construct for each k a circuit Φ_k for $p(G_k)$ satisfying $|\Phi_k|_\times = |I_k| + 1$ and $|\Phi_k|_+ = |I_k| - 1$ simply by using the parenthesizations in the above expressions for $p(G_k)$. Since $\sum_{k=1}^{|H|} I_k = |X_{uw}|$, we have $\sum_{k=1}^{|H|} |\Phi_k|_+ = |X_{uw}| - |H|$ and $\sum_{k=1}^{|H|} |\Phi_k|_\times = |X_{uw}| + |H|$. We can thus obtain a circuit Φ for $p(G)$ by summing the outputs of $\Phi_1, \dots, \Phi_{|H|}$, using $|H| - 1$ \oplus -gates. This construction achieves the desired bounds, which completes the proof of the lemma. \square

5.2 Lower bounds

We now develop the requisite lower bounds for Theorem 5. In Section 3 we showed that optimal circuits for a single path polynomial are monic. The special structure of monic monotone circuits will be key for the proofs in this section. Let Φ be a monic monotone circuit for $p(G)$.

Definition 2 (ζ -unique monomial). *A monomial $A \subseteq X$ is ζ -unique for a node $\zeta \in \Phi$ if there is exactly one monomial $Z \in \text{mon}(\zeta)$ such that $A \subseteq Z$.*

Proposition 5.2. *Let $\rho \in \Phi$ be a \otimes -gate with children α and β . If there is some $A \in \text{mon}(\alpha)$ such that A is ρ -unique, then $|\text{mon}(\beta)| = 1$.*

Proof. If there exist distinct $B_1, B_2 \in \text{mon}(\beta)$, then $A \cup B_1, A \cup B_2 \in \text{mon}(\rho)$, which contradicts the fact that A is ρ -unique. \square

Lemma 5.3. *Let α, ζ be nodes in Φ such that α is a descendant of ζ . If there is some $A \in \text{mon}(\alpha)$ such that A is ζ -unique, then there is a unique path in Φ from α to ζ .*

Proof. We have by Observation 2.4 that Φ is multiplicatively disjoint, and thus two distinct paths in Φ from α to ζ cannot meet at a \otimes -gate. Suppose two such paths meet at a \oplus -gate σ that is a descendant of ζ . It follows that there exist distinct monomials $S_1, S_2 \in \text{mon}(\sigma)$ such that $A \subseteq S_1$ and $A \subseteq S_2$. Then Observation 2.1 implies that there are distinct $Z_1, Z_2 \in \text{mon}(\zeta)$ such that $A \subseteq S_1 \subseteq Z_1$ and $A \subseteq S_2 \subseteq Z_2$, a contradiction. \square

5.2.1 Additive complexity

In this section, we prove a result that is slightly stronger than just the the lower bound in Theorem 5. We will use the fact that every $x \in X_{uw}$ is ϕ -unique to show that every monic monotone circuit Φ for $p(G)$ has, in fact, the *same number* of \oplus -gates.

Lemma 5.4. *If G is a 3-homogeneous st-DAG, then every monic monotone circuit Φ for $p(G)$ satisfies $|\Phi|_+ = |X_{uw}| - 1$.*

Proof. Recall that the monomials in $\text{mon}(\phi)$ are in one-to-one correspondence with the elements of X_{uw} . The following claims establish a notion that is in some sense dual to the notion of a parse tree.

Claim. *If ρ is a \otimes -gate in Φ with children $\{\alpha, \beta\}$, then either $X_{uw} \cap \text{inputs}(\alpha) = \emptyset$ or $X_{uw} \cap \text{inputs}(\beta) = \emptyset$.*

Proof of claim. Suppose there exist indeterminates $x_a, x_b \in X_{uw}$ such that $x_a \in \text{inputs}(\alpha)$ and $x_b \in \text{inputs}(\beta)$ (x_a and x_b must be distinct since Φ is multiplicatively disjoint). Then $\{x_a, x_b\} \subseteq M$ for some $M \in \text{mon}(\rho)$, which is a contradiction as no monomial in $p(G)$ includes more than one indeterminate from X_{uw} . \square

Claim. *If σ is a \oplus -gate in Φ with children $\{\alpha, \beta\}$, then $X_{uw} \cap \text{inputs}(\alpha) \neq \emptyset$ and $X_{uw} \cap \text{inputs}(\beta) \neq \emptyset$.*

Proof of claim. Suppose $X_{uw} \cap \text{inputs}(\alpha) = \emptyset$. Since every monomial in $p(G)$ contains exactly one indeterminate from X_{uw} , there must exist a \otimes -gate ρ in Φ such that (1) ρ has a children $\{\eta, \omega\}$ such that η is an ancestor of σ and ω is not an ancestor of σ ; and (2) $X_{uw} \cap \text{inputs}(\omega) \neq \emptyset$. Since every $x \in X_{uw}$ is ρ -unique, it follows from Proposition 5.2 that $|\text{mon}(\eta)| = 1$. However, the fact that Φ is monic implies $\text{mon}(\alpha) \subset \text{mon}(\sigma)$, thus $|\text{mon}(\sigma)| \geq 2$, which is a contradiction because η is an ancestor of σ . \square

We now construct from Φ a binary tree T_\oplus whose leaves correspond to the elements of X_{uw} and whose internal nodes correspond to the \oplus -gates in Φ . Begin by removing all nodes $\alpha \in \Phi$ for which $X_{uw} \cap \text{inputs}(\alpha) = \emptyset$. Since every node in T_\oplus computes a monomial that is ϕ -unique, T_\oplus is a tree with leaf set X_{uw} . By the latter claim above, every \oplus -gate will remain in T_\oplus , as will both of its children. By the former claim above, at most one child of any \otimes -gate will remain in T_\oplus . We may thus contract every remaining \otimes -gate ρ into its sole remaining child in T_\oplus . Then T_\oplus is a (full) binary tree whose internal nodes are exactly the \oplus -gates in Φ and whose leaves are exactly those inputs from X_{uw} . Since T_\oplus has $|X_{uw}|$ leaves it follows that T_\oplus has exactly $|X_{uw}| - 1$ internal nodes, which completes the proof. \square

5.2.2 Multiplicative complexity

Every monomial of $\text{mon}(\phi)$ is of the form $x_{su_i}x_{u_iw_j}x_{w_jt}$, so we may denote the corresponding parse tree by T_{ij} . In particular, all monomials in $p(G)$ consist of exactly three variables when G is a 3-homogeneous st -DAG, and so each parse tree T_{ij} will contain exactly two \otimes -gates λ_{ij} and v_{ij} such that v_{ij} is an ancestor of λ_{ij} . The three types (I, II, and III) of parse tree for monomials consisting of three indeterminates are shown in Figure 2. Note that every \otimes -gate is either v_{ij} or λ_{ij} for some $i \in \{1, \dots, |U|\}, j \in \{1, \dots, |W|\}$.

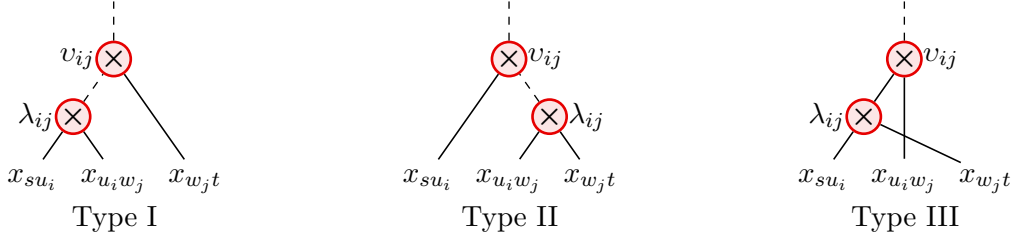


Figure 2: The parse tree $T_{ij} \in \text{PT}(\Phi)$ corresponding to monomial $x_{su_i}x_{u_iw_j}x_{w_jt} \in \text{mon}(\phi)$ must be one of the three types shown here. A dashed edge indicates that one or more \oplus -gates may be present on the path between the nodes indicated. In Lemma 5.5, we show that there is always an optimal monic monotone circuit of minimum size such that no parse tree is of type III.

We establish a canonical form for Φ in the following lemma.

Lemma 5.5. *If Φ is a monic monotone circuit for $p(G)$, then there exists a monic monotone circuit Φ' for $p(G)$ such that $|\Phi'|_{\times} = |\Phi|_{\times}$, $|\Phi'|_{+} = |\Phi|_{+}$, and $x_{u_iw_j}$ is a child of λ_{ij} for all $T_{ij} \in \text{PT}(\Phi')$.*

Proof. We will give an explicit construction for constructing the desired circuit Φ' from Φ .

Let T_{ij} be any parse tree in $\text{PT}(\Phi)$ such that $x_{u_iw_j}$ is not a child of λ_{ij} . Let $\{\alpha, \beta\}$ be the children of v_{ij} , where λ_{ij} is a descendant of α and $x_{u_iw_j}$ is a descendant of β . Now observe that $x_{u_iw_j}$ is ϕ -unique, which, by Proposition 5.2, implies that $|\text{mon}(\beta)| = 1$, thus $\beta = x_{u_iw_j}$. By the same token, it follows from Proposition 5.2 and the fact that $x_{su_i}x_{w_jt}$ is ϕ -unique that $|\text{mon}(\alpha)| = 1$. Since, by Lemma 4.2, all parents of x_{su_i} and x_{w_jt} are \otimes -gates, we may conclude that $\alpha = \lambda_{ij}$ and λ_{ij} computes exactly $x_{su_i}x_{w_jt}$. We have shown that the children of v_{ij} are $\{x_{u_iw_j}, \lambda_{ij}\}$ and the children of λ_{ij} are $\{x_{su_i}, x_{w_jt}\}$. Finally, we note that λ_{ij} is ϕ -unique, which means that v_{ij} must be the sole parent of λ_{ij} by Lemma 5.3. It follows that we may transform Φ in the following way without affecting the polynomial that it computes: (1) create a new \otimes -gate λ_{ij}' with children $\{x_{su_i}, x_{u_iw_j}\}$; (2) create a new \otimes -gate v_{ij}' with children $\{\lambda_{ij}', x_{w_jt}\}$; (3) connect v_{ij}' to the (unique) parent of v_{ij} ; and (4) remove λ_{ij} and v_{ij} from Φ . We may continue with this process to obtain Φ' as desired. Since the number of \otimes -gates and \oplus -gates remains constant throughout the transformation, the proof of the lemma is complete. \square

Lemma 5.6. *If G is a 3-homogeneous st -DAG, then $\mathbf{C}_{\times}(p(G)) \geq |X_{uw}| + \tau(G[U \cup W])$.*

Proof. Let Φ be a monotone circuit for $p(G)$. By Lemma 5.5, we may assume without loss of generality that $x_{u_iw_j}$ is a child of λ_{ij} for all $T_{ij} \in \text{PT}(\Phi)$. Now let $\Lambda = \{\lambda_{ij} \mid T_{ij} \in \text{PT}(\Phi)\}$ and $\Upsilon = \{v_{ij} \mid T_{ij} \in \text{PT}(\Phi)\}$; these sets are disjoint and constitute a partition of the \otimes -gates in Φ . Since each $\lambda_{ij} \in \Lambda$ has $x_{u_iw_j}$ as a child, we have $|\Phi|_{\times} = |X_{uw}| + |\Upsilon|$.

We now demonstrate that $|\Upsilon| \geq \tau(G[U \cup W])$. By our assumption (the one justified by Lemma 5.5), we have that for every parse tree $T_{ij} \in \text{PT}(\Phi)$, λ_{ij} has either x_{su_i} or x_{w_jt} as a child. We can thus construct the desired vertex cover of $G[U \cup W]$ by iterating over the parse trees

$T_{ij} \in \text{PT}(\Phi)$, adding u_i if x_{su_i} is a child of v_{ij} and adding w_j otherwise. The constructed set of vertices constitutes a vertex cover of size $|\Upsilon|$, which completes the proof. \square

6 Concluding Remarks

Recall that our ultimate goal is to determine the complexity of the circuit minimization problem for path polynomials. Specifically, we would like to know whether there is a polynomial-time algorithm that, given any st -DAG G , can build an optimal circuit for $p(G)$. (Broadly speaking, we would like such results for all DAGs, even those with multiple sources and sinks. We defer the challenge of computing collections of polynomials until the computation of a single polynomial is well-understood.)

In this paper, we were able to give some of the first progress in this direction. Specifically, we showed matching upper and lower bounds the additive, multiplicative, and total complexity of $p(G)$ for 3-homogeneous st -DAGs. We also showed the optimal circuits for these polynomials have interesting combinatorial structure. The k -homogeneous st -DAGs are of particular importance for reasons beyond their correspondence to chained matrix products. Observe that the series reduction rule introduced in Section 4 can be re-interpreted as an expansion (rather than a reduction), essentially saying that the act of sub-diving an arc in G will increase the complexity of $p(G)$ by a known, fixed amount in a way that can be subsequently reversed. One implication is that this circuit minimization problem is no harder for general st -DAGs than it is for k -homogeneous st -DAGs.

We note an additional interesting consequence of our results for 3-homogeneous st -DAGs: Let A be a sparse matrix is to be multiplied on the left by a vector x and on the right by a vector y . If the result is to be evaluated by a monotone circuit, then there exist cases where neither bracketing $(xA)y$ nor $x(Ay)$ is optimal. It follows that one need only construct a bipartite graph for which an optimal vertex cover necessarily includes at least one vertex from each of the two vertex sets for this to be so.

Acknowledgments

The author wishes to thank Amit Chakrabarti, Andreas Griewank, Uwe Naumann, Alexander Razborov, and Jean Utke for numerous discussions related to the material presented here. This work was supported in part by the U.S. Department of Energy, under Contract DE-AC02-06CH11357.

References

- [1] E. Allender, J. Jiao, M. Mahajan, and V. Vinay. Non-commutative arithmetic circuits: depth reduction and size lower bounds. *Theoretical Computer Science* 209(1-2):47 – 86, 1998, doi:10.1016/S0304-3975(97)00227-2. 6
- [2] A. Griewank and U. Naumann. Accumulating Jacobians as chained sparse matrix products. *Mathematical Programming* 3(95):555–571, 2003, doi:10.1007/s10107-002-0329-7. 3
- [3] A. Griewank and A. Walther. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. Other Titles in Applied Mathematics 105. SIAM, Philadelphia, 2nd edition, 2008. 2
- [4] T. C. Hu and M. T. Shing. Computation of matrix chain products. Part I. *SIAM Journal on Computing* 11(2):362–373, 1982, doi:10.1137/0211028. 3

- [5] T. C. Hu and M. T. Shing. Computation of matrix chain products. Part II. *SIAM Journal on Computing* 13(2):228–251, 1984, doi:10.1137/0213017. 3
- [6] M. Iri. Simultaneous computation of functions, partial derivatives, and estimates of rounding errors —complexity and practicality—. *Japan Journal of Applied Mathematics* 1(2):223–252, 1984, doi:10.1007/BF03167059. 3
- [7] M. Jerrum and M. Snir. Some exact complexity results for straight-line computations over semirings. *Journal of the ACM* 29(3):874–897, 1982, doi:10.1145/322326.322341. 3, 4, 5, 6
- [8] B. Mahr. A birds eye view to path problems. *WG*, pp. 335–353. Springer, Lecture Notes in Computer Science 100, 1980, doi:10.1007/3-540-10291-4_25. 3
- [9] B. Mahr. Algebraic complexity of path problems. *Informatique Théorique* 16(3):263–292, 1982. 1, 3
- [10] K. Mehlhorn. *Data Structures and Algorithms 2: Graph Algorithms and NP-Completeness*. Monographs in Theoretical Computer Science, an EATCS Series 2. Springer, 1984, <http://www.mpi-sb.mpg.de/~mehlhorn/DatAlgbooks.html>. 3
- [11] N. Nisan and A. Wigderson. Lower bounds on arithmetic circuits via partial derivatives. *Computational Complexity* 6(3):217–234, 1997, doi:10.1007/BF01294256. 6
- [12] R. Raz and A. Yehudayoff. Lower bounds and separations for constant depth multilinear circuits. *Computational Complexity* 18(2):171–207, 2009, doi:10.1007/s00037-009-0270-8. 6
- [13] V. Strassen. Gaussian elimination is not optimal. *Numerische Mathematik* 13(4):354–356, 1969, doi:10.1007/BF02165411. 2
- [14] V. Vassilevska. *Efficient Algorithms for Path Problems in Weighted Graphs*. Ph.D. thesis, Carnegie Mellon University, August 2008, <http://reports-archive.adm.cs.cmu.edu/anon/2008/CMU-CS-08-147.pdf>. 1
- [15] R. Yuster and U. Zwick. Fast sparse matrix multiplication. *ACM Trans. Algorithms* 1(1):2–13, 2005, doi:10.1145/1077464.1077466. 1