IEICE TRANS. INF. & SYST., VOL. E75-D, NO. 1 JANUARY 1992

PAPER Special Section on Theoretical Foundations of Computing

Classes of Arithmetic Circuits Capturing the Complexity of Computing the Determinant

Seinosuke TODA†, *Member*

SUMMARY In this paper, some classes of arithmetic circuits are introduced that capture the computational complexity of computing the determinant of matrices with entries either indeterminates or constants from a field. An *arithmetic circuit is* just like a Boolean circuit, except that all AND and OR gates (with fan-in two) are replaced by gates realizing a multiplication and an addition, respectively, of two polynomials over some indeterminates with coefficients from the field, and the circuit computes a (formal multivariate) polynomial in the obvious sense. An arithmetic circuit is said to be *skew* if at least one of the inputs of each multiplication gate is either an indeterminate or a constant. Then it is shown that for all square matrices M of dimension q, the determinant of M can be computed by a skew arithmetic circuit of $\mathcal{O}(q^{20})$ gates, and is shown that for all skew arithmetic circuits C of size q, the polynomial computed by C can be defined as the determinant of a square matrix M of dimension $\mathcal{O}(q)$. Thus the size of skew arithmetic circuit is polynomially related to the dimension of square matrices when it is considered to represent multivariate polynomials in both arithmetic circuits and the determinant. The results are extended to some other classes of arithmetic circuits less restricted than skew ones, and by using such an extended result, a difference and a similarity are demonstrated between polynomials represented as the determinant of matrix of relatively small dimension and those polynomials computed by arithmetic formulas and arithmetic circuits of relatively small size and degree.

kew words: *complexity Theory, arithmetic circuit, determinant*

1. Introduction

It is well understood that the computational complexity of computing the determinant captures the complexity of most elementary computations in linear algebra, such as inverting matrices, computing the coefficients of the characteristic polynomial, computing the power series of a given matrix, and computing the rank of a given matrix, and hence it has been important to investigate the complexity of computing the determinant. In spite of its importance, its precise complexity have remained to be characterized in terms of some suitable computational models of computation; that is, no complexity class defined by a reasonable model of computations has been known for capturing the computational power of the determinant. Such a question has been initiated by Valiant[3] and by Cook[2] in two different ways. Valiant considered an algebraic setting based on arithmetic formulas and on arithmetic circuits, and observed a universality of the determinant in the sense that all the formal polynomial computed by arithmetic formulas can be represented efficiently as the determinant, where by "efficient" we mean the dimension of a matrix concerned is polynomially related to the length of the formula. Cook has defined the class of functions (alternatively, sets) that are NC^1-reducible to computing the determinant of integer matrices, and exhibited some complete functions for the class under NC^1-reducibility. Nonetheless, they have not given a satisfactory characterization in their contexts to the power of the determinant.

Motivated with this situation, we investigate the nature of what the determinant compute, from the complexity theoretic viewpoint. In particular, we are concerned with two purposes: characterizing the complexity of computing the determinant in terms of arithmetic circuits and demonstrating a difference and/or a similarity to the other classes of polynomials computed by arithmetic circuits. To accomplish the purposes, we will introduce some classes of arithmetic circuits that capture the complexity of computing the determinant.

An *arithmetic circuit* is just like a Boolean circuit, except that all AND and OR gates (with fan-in two) are replaced by gates realizing a multiplication and an addition, respectively, of two polynomials over some indeterminates with coefficients from a field. Given an arithmetic circuit C, we can define the (formal multivariate) polynomial computed by C in the obvious way. An arithmetic circuit is said to be *skew* if at least one of the inputs of each multiplication gate is either an indeterminate or a constant. Then we show that for all square matrices M of dimension q with entries either indeterminates or constants from a field, the determinant of M can be computed by a skew arithmetic circuit of $\mathcal{O}(q^{20})$ gates, and show that for all skew arithmetic circuits C of size q, the polynomial computed by C can be defined as the determinant of a square matrix M of dimension $\mathcal{O}(q)$. The results are extended to some other classes of arithmetic circuits less restricted than skew ones. An arithmetic circuit is called *weakly skew* if for every multiplication gate in the circuit, at least one of two subcircuits for the inputs

Manuscript received May 1, 1991.
Munuscript revised September 25, 1991.

† The author is with the Faculty of Electro-Communications, University of Electro-Communications, Chofu-shi, 182 Japan.

of the gate is used only for the multiplication gate; that is, there is no connection line from the subcircuit to other gates in the remaining part of the circuit. Notice that this notion generalizes the notion of skew arithmetic circuits. Then we show that the results mentioned above remain true for the weakly skew arithmetic circuits. We will show some similar results for some other classes.

In the context of arithmetic circuits, a few results have been known on the complexity of computing the determinant. As roughly mentioned above, it was shown by Valiant[3] that an arithmetic formula containing s operators can be defined as the determinant of a square matrix of dimension $s+2$. It is known that the determinant can be computed by a family of arithmetic circuits of relatively small size and degree (for these notions see the next section). Though the latter result was not explicitly presented in the literature, it follows from translating the determinant algorithm developed by Berkowitz[1] into arithmetic circuits. Thus arithmetic formulas and arithmetic circuits of relatively small size and degree may be considered to give a lower bound and an upper bound, respectively, on the complexity of computing the determinant in the context of arithmetic circuits. To compare these results with ours, we will see, from the definition, that the class of weakly skew arithmetic circuits naturally extends the class of arithmetic formulas and are a subclass of arithmetic circuits of relatively small degree. Finally, we will demonstrate a difference and a similarity among those formulas, circuits, and the determinant, by using the result extended to weakly skew arithmetic circuits; that is, we will observe that an arithmetic circuit for computing the determinant uses multiplication gates just as in arithmetic formulas but uses addition gates just as in the degree bounded case.

2. Preliminaries

In this section we define some necessary notions and notations on arithmetic circuits. We assume familiarity with basic notions on graphs, linear algebra, and computational complexity. Throughout this paper, $\mathcal{F}$ and $\mathcal{X}$ respectively denote a field and the set of countably infinite indeterminates $\{x_i \mid i \geq 1\}$. $\mathcal{F}[x_1, x_2, \cdots, x_n]$ denotes the ring of polynomials over indeterminates $x_1, \cdots, x_n$ with coefficients from $\mathcal{F}$.

Definition 2.1: An *arithmetic circuit* C (over $\mathcal{F}$ with n indeterminates) is a sequence of instructions of the form $v_i := u_i \circ w_i$ for $i=1, 2, \cdots, m$, where

(1) $u_i, w_i \in \mathcal{F} \cup \{x_1, \cdots, x_n\} \cup \{v_1, \cdots, v_{i-1}\}$, and

(2) $\circ$ is one of the two ring operators $+$, $\times$ over $\mathcal{F}[x_1, \cdots, x_n]$.

Note that since $-1 \in \mathcal{F}$, subtraction can be easily realized in our arithmetic circuits. The (*formal multivariate*) polynomial computed at v_i can be defined in the obvious way and is denoted $P(C; v_i)$. The *polynomial* computed by C is defined as $P(C; v_m)$ and is simply denoted $P(C)$.

An arithmetic circuit may be redundant; that is, it may contain an instruction that is never used for computing the objective polynomial. Throughout the present paper, we assume that any circuits are not redundant. That is, our circuits are minimal in the sense that any proper subset of the instructions cannot compute the same polynomial as the original one.

Though we define an arithmetic circuit as a set of instructions as above, we can alternatively define it as an acyclic digraph in a usual way. We here give the formal definition of arithmetic circuits in this way in order to make it clear how the former definition relates to the latter. The *underlying graph* of the circuit C is an acyclic node-labeled digraph (V_C, E_C) defined as follows:

(1) $V_C = \{u_i \mid 1 \leq i \leq m\} \cup \{x_1, \cdots, x_n\} \cup \{d \in \mathcal{F} \mid d$ appears in $C\}$. Each node v_i in the first set above is of label $+$ (resp., $\times$) if the ith instruction is an addition (resp., a multiplication) one. All nodes in the second and the third sets above are labeled with themselves.

(2) For all three nodes v, u, and w where v is in the first set above, the instruction $v := u \circ w$ is in C iff the edges (u, v) and (w, v) are in E_C, where $\circ$ denotes the label of v in C.

We will denote (V_C, E_C) simply by C itself unless otherwise specified. We call each node a *gate* of C and call v_m the *output gate of* C. We conventionally call an x_i (and a d) above an *input gate* (resp., a *constant gate*) of C. By a *multiplication gate* (an *addition gate*), we mean a gate with label $\times$ (resp., $+$). The size of the circuit is m above., i.e., the number of the instructions (alternatively, the number of all gates except input and constant ones). The *degree at a gate* v_i *of* C is defined to be the degree of $P(C; v_i)$. We define the *degree of the circuit* to be that of $P(C)$.

Remark: We will use one of the two definitions of arithmetic circuits according to the context.

Definition 2.2: Intuitively, an *arithmetic formula* (over a field $\mathcal{F}$ with n indeterminates) is an arithmetic circuit such that every gate except input gates and constant gates appears exactly once in the right hand side of an instruction. More formally, an arithmetic formula is inductively defined as follows:

(1) all indeterminates in $\mathcal{X}$ and all constants in $\mathcal{F}$ are arithmetic formulas,

(2) if f and g are arithmetic formulas, $(f \circ g)$ is an arithmetic formula, where $\circ$ is one of the two ring operators over $\mathcal{F}[\mathcal{X}]$, and

(3) what is constructed above is all of the arithmetic formulas.

The *size* of an arithmetic formula is the number of the ring operators appearing in the formula, which equals the number of instructions contained in an arithmetic

circuit into which the formula can be translated in the obvious way.

We define some classes of families of polynomials in $\mathcal{F}[\mathcal{X}]$.

Definition 2. 3: By a *family of polynomials* we mean a countably infinite sequence $\mathcal{P}=\{P_n\}$, $n\geqq 1$ of polynomials in $\mathcal{F}[\mathcal{X}]$ such that for all $n\geqq 1$, P_n is a polynomial in $\mathcal{F}[x_1, x_2, \cdots, x_n]$. By a *family of arithmetic circuits* we mean a countably infinite sequence $\mathcal{C}=\{C_n\}$, $n\geqq 1$ of arithmetic circuits such that for all $n\geqq 1$, all indeterminates appearing in C_n are in $\{x_1, \cdots, x_n\}$. We say that $\mathcal{C}$ computes $\mathcal{P}$ if for all $n\geqq 1$, C_n computes P_n.

We define ARC-SIZE, DEG(poly, poly) to be the class of all families of polynomials computed by families of arithmetic circuits $\{C_n\}$, $n\geqq 1$ such that for some polynomial $s(\cdot)$ and all $n\geqq 1$, C_n is of both size and degree at most $s(n)$. We define ARF-SIZE(poly) to be the class of all families of polynomials computed by families of arithmetic formulas $\{F_n\}$, $n\geqq 1$ such that for some polynomial $s(\cdot)$ and all $n\geqq 1$, F_n is of size at most $s(n)$.

We define one more class of families of polynomials, which plays a central role in the present paper.

Definition 2. 4: Let M be a square matrix whose entries each is either a constant from $\mathcal{F}$ or an indeterminate. By $\det(M)$ we denote the determinant of M. Note that $\det(M)$ defines a formal polynomial in $\mathcal{F}[\mathcal{X}]$. By a *family of matrices,* we mean a countably infinite sequence of square matrices $\{M_n\}$, $n\geqq 1$ such that for all $n\geqq 1$, all indeterminates appearing in M_n are in $\{x_1, x_2, \cdots, x_n\}$. Then we define DET(poly) to be a class of all families of polynomials $\{P_n\}$, $n\geqq 1$ for which there exists a family of matrices $\{M_n\}$, $n\geqq 1$ and a polynomial $s(\cdot)$ such that for all $n\geqq 1$, $P_n=\det(M_n)$ and M_n is of dimension at most $s(n)$.

It is currently known that for a square matrix M of dimension s, $\det(M)$ can be computed by an arithmetic circuit of size and degree at most s^k for a fixed constant $k\geqq 0$ independent of the dimension of M. Though the relationship was not explicitly observed in any papers, it follows from an algorithm developed by Berkowiz[1] for computing the determinant. We will also use his idea for proving our main result. It is also known that a polynomial computed by an arithmetic formula of size s can be defined as the determinant of a square matrix of dimension $s+2$. This relationship was shown by Valiant[3]. By these relationships, we see ARF-SIZE(poly) $\subseteq$ DET(poly) $\subseteq$ ARC-SIZE, DEG (poly, poly). It currently remains open whether one of the two inclusions is proper or not; that is, it remains open whether ARF-SIZE(poly) $\neq$ DET(poly) or DET (poly) $\neq$ ARC-SIZE, DEG(poly, poly). Valiant[3] seemed to be first aware of the questions and the latter was explicitly posed in Ref.(4). Nonetheless, at the present time, it seems hard to settle one of the questions even if one of the equalities holds (if one of the inequalities could be proved, then we might be able to show a proper inclusion among some important complexity classes such as DL (Deterministic Log Space) and PTIME (Deterministic Polynomial Time)). Thus our interest in this paper is to find a good class of families of arithmetic circuits that captures DET(poly) exactly, rather than to settle the questions, and is to demonstrate differences and/or similarities among the above three classes by such results.

3. Skew Arithmetic Circuits and the Determinant

In this section, we first define the class of *skew arithmetic circuits* and show that the class precisely captures the computational complexity of computing the determinant. A notion of skew circuits in the context of Boolean circuits were defined by Venkateswaran[5]. Our notion below is a natural translation of his one into the context of arithmetic circuits.

Definition 3. 1: A gate of an arithmetic circuit is said to be *skew* if at least one of its inputs is either an indeterminate or a constant. An arithmetic circuit is said to be *skew* if every multiplication gate is skew. By SkewARC-SIZE(poly) we denote the class of all families of polynomials computed by families of skew arithmetic circuits $\{C_n\}$, $n\geqq 1$ such that for some polynomial $s(\cdot)$ and all $n\geqq 1$, C_n is of size at most $s(n)$.

We can easily see, by induction on the number of instructions, that the degree of a skew arithmetic circuit is bounded above by the number of gates. Thus we see SkewARC-SIZE(poly) $\subseteq$ ARC-SIZE,DEG(poly, poly). Thought it is not obvious whether SkewARC-SIZE(poly) includes ARF-SIZE(poly), our main result below tells us that it is the case.

Theorem 3. 2: SkewARC-SIZE(poly) = DET(poly).

The theorem follows from Lemma 3. 4 and Lemma 3. 5 below. We first show a technical lemma used for proving Lemma 3. 4.

Lemma 3. 3: Let W be a square matrix of dimension m whose entries each is either a constant or an indeterminate. For all $1\leqq i, j\leqq m$, let P_{ij} denote the polynomial that is obtained as the (i, j)th entry of $\sum_{k=1}^{m} W^k$. Then, there exists a skew arithmetic circuit C such that its size is $\mathcal{O}(m^4)$ and for all $1\leqq i, j\leqq m$, P_{ij} is computed at a gate v_{ij} of the circuit.

Proof: We can construct a skew arithmetic circuit D that computes W^m, simply according to the inductive definition of matrix powering. Then we see that for all $1\leqq i, j, k\leqq m$, the polynomial obtained as the (i, j)th entry of W^k is computed at a gate v_{ij}^k of D. By adding to D some new addition gates for computing $\sum_{k=1}^{m} P(D; v_{ij}^k)$ for all $1\leqq i, j\leqq m$, we obtain a circuit C that computes each P_{ij} at some gate. Note that the resulting

circuit remains skew. The size of D is at most $m^2(2m-1)(m-1)$, and $m^2(m-1)$ addition gates are used for taking the sums. Thus we have the desired size bound. □

Lemma 3.4: Let A be a square matrix of dimension q whose entries each is either a constant or an indeterminate. Then there is a skew arithmetic circuit C of size $\mathcal{O}(q^{20})$ such that $P(C)=\det(A)$.

Proof: The proof is based on an algorithm computing the determinant of the matrices, which was developed by Berkowitz[1]. Below, we first outline his algorithm for the completeness of this proof; the reader may refer to his paper for its correctness. Let $1\leqq t\leqq q$. We define $A[t]$ to be a $(q-t+1)\times(q-t+1)$ matrix obtained by deleting, for all $1\leqq i<t$, the ith row and the ith column from A. We define $R[t]$ and $S[t]$ as $1\times(q-t)$ and $(q-t)\times 1$ matrices obtained from the first row and column of $A[t]$, respectively, by deleting each first entry. We also define $M[t]$ as a $(q-t)\times(q-t)$ matrix obtained by deleting the first row and the first column of $A[t]$. The relationship among $A[t]$, $R[t]$, $S[t]$, and $M[t]$ is illustrated as follows:

$$A[t]=\begin{bmatrix} A_{t,t} & R[t] \\ S[t] & M[t] \end{bmatrix},$$

where $A_{t,t}$ denotes the (t, t)th entry of A. Let $(p_0, \cdots, p_q)$ be the coefficients of the characteristic polynomial of A; that is,

$$\sum_{i=0}^{q} p_i \cdot \lambda^{q-i}=\det(A-\lambda \cdot I),$$

where I denotes the $q\times q$ unit matrix. Thus p_q is the determinant of A. We say that a matrix is *Toeplitz* if each diagonal has a value to which all the entries on the diagonal are equal. Using these notations and notions, his algorithm can be described as follows:

Step 1: For each $1\leqq t\leqq q$, compute a $(q-t+2)\times(q-t+1)$ matrix $C[t]$ that are lower triangular and Toeplitz and are defined as follows:

$$C[t]=\begin{bmatrix} -1 & & & & & & \\ a & -1 & & & & & \\ v_3 & a & -1 & & & & \\ v_4 & v_3 & a & -1 & & & \\ \vdots & \vdots & v_3 & & & & \\ v_{k-2} & \vdots & \vdots & & a & -1 & \\ v_{k-1} & v_{k-2} & \vdots & & v_3 & a & -1 \\ v_k & v_{k-1} & v_{k-2} & & v_4 & v_3 & a \end{bmatrix}$$

where $k=q-t+2$, $a=A_{t,t}$, and for each $3\leqq l\leqq k$, $v_l=-R[t]\cdot M[t]^{l-3}\cdot S[t]$.

Step 2: Compute $(c_0, c_1, \cdots, c_q)^T=\prod_{t=1}^{q} C[t]$, and output c_q, where $(\cdots)^T$ means to transpose the matrix.

For this algorithm, Berkowitz showed that $(c_0, \cdots, c_q)$ above equals $(p_0, \cdots, p_q)$; hence the output above is the determinant of A. Now we give a skew arithmetic circuit that computes the determinant of A. In order to give it, we will construct an acyclic digraph G such that each edge is of weight either an entry of A or -1, and we will observe that an entry of the transitive closure of the weight matrix of G gives the determinant of A. Combining the observation with Lemma 3.3, we will obtain this lemma.

To construct G, we first construct a digraph $G_1=(V_1, E_1)$ as follows:

(1) $V_1=\{v_i^t \mid 1\leqq t\leqq q+1,\ 1\leqq i\leqq q-t+2\}$.

(2) For each $1\leqq t\leqq q$, $1\leqq i\leqq q-t+2$, and $1\leqq j\leqq q-t+1$, if $C[t]_{i,j}$ is either -1, $A_{t,t}$, or the product $-R[t]\cdot M[t]^{l-3}\cdot S[t]$ (for some $3\leqq l\leqq q-t+2$), then G has the edge (v_i^t, v_j^{t+1}) with weight -1, $A_{t,t}$, or the product respectively; otherwise, G has no edge between v_i^t and v_j^{t+1}.

It is easy to see that c_q is given by Σ_p {product of all weights on p} where the summation is taken over all simple paths in G_1 from v_{q+1}^1 to v_1^{q+1}. Note that G_1 contains edges with weight of the form $-R[t]\cdot M[t]^{l-3}\cdot S[t]$, which is in general neither an entry of A nor -1. For the sake of eliminating such all edges while keeping the fact on c_q mentioned above, we below consider to replace each of such all edges to a digraph in which all weights are either entries of A or -1. For $1\leqq t\leqq q-1$ and $3\leqq l\leqq q-t+2$, we define a digraph $H^{t,l}=(V^{t,l}, E^{t,l})$ as follows:

(1) $V^{t,l}=\{r_1, r, s\}\cup\{m_i^h \mid 1\leqq h\leqq l-2,\ 1\leqq i\leqq q-t\}$.

(2) E contains the following edges:

(a) (r_1, r_2) with weight -1.

(b) (r_2, m_i^1) with weight $R[t]_{1,i}$ for all $1\leqq i\leqq q-t$.

(c) (m_i^h, m_j^{h+1}) with weight $M[t]_{i,j}$ for all $1\leqq h\leqq l-3$ and $1\leqq i, j\leqq q-t$.

(d) (m_i^{l-2}, s) with weight $S[t]_{i,1}$ for all $1\leqq i\leqq q-t$.

It is easy to see that the product $-R[t]\cdot M[t]^{l-3}\cdot S[t]$ is given by Σ_p {product of all weights on p} where the summation is taken over all simple paths in $H^{t,l}$ from r_1 to s. For each edge (v_i^t, v_j^{t+1}) in G_1 that has weight of the form $-R[t]\cdot M[t]^{l-3}\cdot S[t]$ for some $1\leqq l\leqq q-t+2$, we replace the edge to a copy of $H^{t,l}$ by identifying v_i^t and v_j^{t+1} with r_1 and s respectively. G is the resulting digraph. From the observations on G_1 and $H^{t,l}$ mentioned above, we have that c_q is given by Σ_p {product of all weights on p} where the summation is taken over all simple paths in G from v_{q+1}^1 to v_1^{q+1}.

Let m be the number of nodes in G and let's suppose that all nodes in G are indexed by numbers between 1 through m. We also suppose that v_{q+1}^1 and v_1^{q+1} are the first node and the mth node, respectively. Then we define W as a square matrix of dimension m such that the (i, j)th entry of W is the weight of the edge of G connecting from the ith node to the jth

node; if such an edge does not exists, then the entry is zero. Then we can obtain c_q in Step 2 above as the $(1, m)$th entry of $\sum_{i=1}^{m} W^i$. Thus, by Lemma 3. 3, we obtain a skew arithmetic circuit of size $\mathcal{O}(m^4)$ that computes the determinant of A. It remains to estimate the size of the resulting circuit. The number of nodes in G_1 is $\mathcal{O}(q^2)$, and for all $1 \leqq t \leqq q-1$ and $1 \leqq l \leqq q-t+2$, the number of nodes in $H^{t,l}$ is $\mathcal{O}(q^2)$. We use at most $|E_1|$ copies of $H^{t,l}$'s. Thus the total number of nodes in G is $\mathcal{O}(q^2)+\mathcal{O}(q^2) \cdot |E_1| = \mathcal{O}(q^5)$. By Lemma 3. 3, the size of the resulting circuits is $\mathcal{O}(q^{20})$. □

Lemma 3. 5: Let C be a skew arithmetic circuit of size q. Then there exists a square matrix M of dimension at most $4q+3$ such that $P(C) = \det(M)$.

Proof: Our first task before constructing a desired matrix is to construct, from the circuit C, an acyclic digraph whose edges are labeled with either constants or indeterminates as their weights. The digraph satisfies that for two specified nodes s and t, $P(C)$ is given by $\Sigma_p\{$product of all weights on $p\}$ where the summation is taken over all simple paths in the digraph from s to t. After that, we will construct the matrix from the digraph. The digraph is constructed as follows.

[Graph Construction] Let G be the underlying digraph of C.

(For multiplication gates) Let v be a node of G that represents a multiplication gate, and let u be a node of G that is labeled with an indeterminate x (or a constant c) and is an input for the multiplication gate. Then we delete the edge (u, v) from G, and we assign weight x (resp., weight c) to all edges coming out from v. If u become an isolated node after deleting the edge (u, v), then we delete u from G.

(For addition gates) We do nothing for all nodes that represent addition gates.

(For constant and input gates) For each node v that represents a constant c (or an indeterminate x), we assign weight x (resp., weight c) to all edges coming out from v.

(Some remaining processes) In the digraph obtained so far, we assign weight one to all edges remaining unweighted by the above process. We introduce a new node s and introduce an edge with weight one from s to each node that represents a constant or an indeterminate.

We denote by G_1 the digraph obtained above and denote by t the node in G_1 that corresponds to the output gate of C. Next we define an edge-weighted digraph G_2, which is obtainced from G_1 as follows.

(1) For each node v in G_1 except s:
  (a) we introduce two new nodes v_1 and v_2 and introduce the edge (v_1, v_2) with weight one,
  (b) for all edges (w, v) in G_1 going into v, we introduce the edge (w, v_1) with the same weight as (w, v),
  (c) for all edges (v, w) in G_1 coming out from v, we introduce the edge (v_2, w) with the same weight as (v, w), and
  (d) we delete the node v and all edges incident with v from G_1.
(2) We add the edge (t_2, s) with weight one to the digraph obtained above, where t_2 denotes the node introduced in (a) above for the node t in G_1.
(3) Finally, we attach a self-loop to each node except s, where all self-loops are of weight one. G_2 is the resulting digraph.

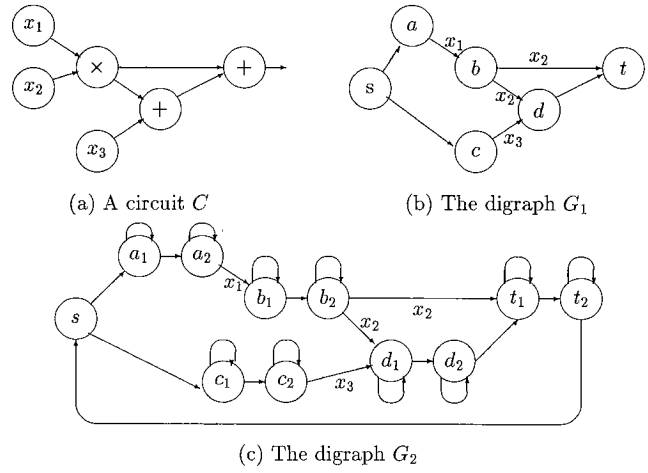


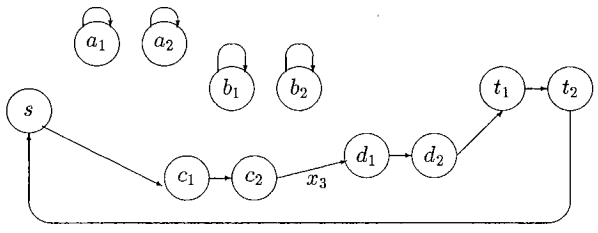
Fig. 1 An example of the constructions in Lemma 3. 3, where all unlabeled edges have weight one.

Note that all simple cycles in G_2 containing s are of odd length, where the length of a cycle is the number of nodes on the cycle. We will use this fact later.

Before defining the matrix M, we state some facts on G_1 and G_2 that will be used later. For simplicity, we call a simple path (in G_1 or G_2) from the node s to the node t an (s, t)-path, and call a simple cycle (in G_2) containing s an s-cycle, where a *simple* path (a *simple* cycle) means a path (resp., a cycle) that passes through each node at most once. Below, note that a simple cycle may be in general a self-loop but any s-cycle cannot be a self-loop since s has no self-loop in G_2.

From the definition of G_1, we easily see that $P(C)$ is given by $\Sigma_p\{$product of all weights on $p\}$ where the summation is taken over all (s, t)-paths in G_1. From the definition of G_2, we can easily see a one-to-one correspondence between all (s, t)-paths in G_1 and all s-cycles in G_2 such that the product of all weights on an (s, t)-path in G_1 equals the product of all weights on the corresponding s-cycle in G_2. Thus we have that $P(C)$ is given by $\Sigma_c\{$product of all weights on $c\}$ where the summation is taken over all s-cycles in G_2.

Let c be any s-cycle in G_2. We consider a set δ consisting of all edges on c and self-loop edges of all nodes not appearing in c (Here by an edge we mean a



$$\pi_\delta=\begin{pmatrix} s & a_1 & a_2 & b_1 & b_2 & c_1 & c_2 & d_1 & d_2 & t_1 & t_2 \\ c_1 & a_1 & a_2 & b_1 & b_2 & c_2 & d_1 & d_2 & t_1 & t_2 & s \end{pmatrix}$$

Fig. 2 An example of δ in Δ and its corresponding permutation π_δ (see Lemma 3.5), where all unlabeled edges have weight one.

pair of the edge and its weight). Let Δ denote the family of all such δ's. Then we see a natural one-to-one correspondence between all s-cycles in G_2 and Δ, and see that the product of all weights on an s-cycle in G_2 equals the product of all weights appearing in its corresponding δ in Δ. Thus we have $P(C)=\Sigma_{\delta\in\Delta}\{\text{product of all weights appearing in } \delta\}$. One more important fact on G_2 is that each s-cycle is of odd length.

Now we are ready to define the matrix M. Let h be the number of nodes in G_2, and let's assume that all nodes in G_2 are indexed by numbers between 1 through h, and s and t are of the index 1 and h, respectively. Then, we define M as a $h\times h$ matrix such that for each $1\leqq i, j\leqq h$, its (i, j)th entry is the weight of the edge in G_2 connecting from the ith node to the jth node; if such an edge does not exists, then the entry is zero. To show $P(C)=\det(M)$, it is sufficient to show $\det(M)=\Sigma_{\delta\in\Delta}\{\text{product of all weights in } \delta\}$. For that purpose, we assume that the circuit C does not contain zero as its constant gate. We can assume so without loss of generality. From this assumption, we have that all egdes in G_2 have non-zero weight and hence, for all $\delta\in\Delta$, the product of all weights in δ is not zero. Let Γ be the set of all permutations π of h elements such that its corresponding monomial in $\det(M)$, denoted $m(\pi)$, is not zero (hence, $\det(M)=\Sigma_{\pi\in\Gamma}m(\pi)$). Below, we show that there exists a one-to-one correspondence between Δ and Γ such that for all $\delta\in\Delta$, the product of all weights in δ equals $m(\pi_\delta)$, where π_δ denotes the permutation in Γ corresponding to δ. At Fig. 2, we given an example of δ in G_2 of Fig. 1 and its corresponding permutation.

Let $\delta\in\Delta$. Note that in the subgraph induced by δ, each node is of indegree one and outdegree one. Hence, when we define π_δ as a function on $\{1, 2, \cdots, h\}$ such that for all $1\leqq i\leqq h$, $\pi_\delta(i)$ is the index of the successor of the ith node in the subgraph, we see that π_δ is a permutation of h elements and the product of all weights in δ equals $\prod_{i=1}^{h}M_{i,\pi_\delta(i)}$, where $M_{i,\pi_\delta(i)}$ denotes the $(i, \pi_\delta(i))$th entry of M. Furthermore, we see that π_δ is a cyclic permutation of odd length, because the digraph defined by the edge set $\{(i, \pi_\delta(i)) \mid 1\leqq i\leqq h\}$ is isomorphic to the subgraph above that consists of an s-cycle in G_2 of odd length and some self-loops. Note that the parity sign of any cyclic permutation of odd length is $+1$. Hence, we have $m(\pi_\delta)=sgn(\pi_\delta)\cdot\prod_{i=1}^{h}M_{i,\pi_\delta(i)}=\prod_{i=1}^{h}M_{i,\pi_\delta(i)}=$the product of all weights in δ. This also implies $\pi_\delta\in\Gamma$. It is easy to see that the mapping from Δ to Γ defined as above is an injection.

To show the converse correspondence, let $\pi\in\Gamma$. Define $\delta_\pi=\{(v_i, v_{\pi(i)}) \mid 1\leqq i\leqq h\}$, where v_i denotes the ith node in G_2. Since $m(\pi)$ is not zero and hence each $M_{i,\pi(i)}$ is not zero, all elements in δ_π are edges in G_2. It is obvious that the product of all weights on edges in δ_π equals $\prod_{i=1}^{h}M_{i,\pi(i)}$. Below, we show that the subgraph induced by δ_π consists of an s-cycle and self-loops of all nodes not appearing in the s-cycle, which implies $\delta_\pi\in\Delta$. Since π is a permutation, each node in the subgraph is of indegree one and outdegree one. This means that the subgraph consists of some mutually node-disjoint simple cycles. Assume the subgraph contains at least two simple cycles which are not self-loops. Then, from the definition of G_2, those simple cycles must pass through the node s. This contradicts that those simple cycles are mutually node-disjoint. Assume the subgraph contains only self-loops. Then the node s must have a self-loop. This is also a contradiction. Thus the subgraph consists of one simple cycle not being self-loop and self-loops of nodes not appearing on the simple cycle, and it is obvious from the definition of G_2 that the simple cycle must be an s-cycle. Thus we have $\delta_\pi\in\Delta$. We easily see that the mapping from Γ to Δ defined as above is an injection and is the inverse of the previous mapping from Δ to Γ. Since any s-cycle must be of odd length, we see that π is a cyclic permutation of odd length, and hence, we have that the product of all weights on edges in δ_π equals $m(\pi)$.

From this one-to-one correspondence, we have $\det(M)=\Sigma_{\delta\in\Delta}\{\text{product of all weights in } \delta\}=P(C)$. It remains to estimate h, the dimension of M, which is also the number of nodes in G_2. The number of nodes in G_1 is at most $2q+2$, since at least $q-1$ arguments of the instructions in C must be internal gates and hence the number of constants and indeterminate gates in C is at most $q+1$ (recall that we do not include the number of the input gates in the size of circuits and all our circuits are not redundant). Hence, we see, from the definition of G_2, that h is at most $4q+3$. This completes the proof. □

4. Extensions of Skew Arithmetic Circuits

The structure of the underlying graphs of skew arithmetic circuits seems too restricted, while it is

simple and easy to understand. In particular, we cannot see, immediately from their structure, the fact that skew arithmetic circuits can simulate all arithmetic formulas with polynomial increase in size. Motivated by this, we here introduce a class of arithmetic circuits which are of more loose restriction in the syntax than skew ones and obviously subsumes arithmetic formulas and skew arithmetic circuits, and we show that Theorem 3. 2 remains true for the class here. Later in this section, we will introduce two natural reducibility notions between families of multivariate polynomials, called *arithmetic formula reducibility* and $NC^1_{\mathcal{F}}$-reducibility, and will consider the downward closures of DET(poly) under the reducibilities. Those closures can be viewed as algebraic analogues to DET defined by Cook[(2)] as the class of functions over binary strings computed by NC^1 circuits with oracle gates for the determinant of integer matrices where all integers are represented in binary. We will observe, by using the first result of this section, that both closures are equal to DET(poly) itself, that is, DET(poly) is closed downward under both reducibilities.

The definition of the class introduced here is based on the notion of subcircuits. We first give the formal definition.

Definition 4. 1: Let C be an arithmetic circuit and v be a gate of C. A *subcircuit for* v (in C) is a subset of instructions of C that computes the same polynomial as computed at v. A subcircuit may contain some redundant instructions. We say a subcircuit D for v is *minimal* if all proper subcircuits of D cannot compute the same polynomial as D does. Notice that a minimal subcircuit for v is uniquely determined in the obvious way. Hereafter, when we talk about subcircuits of a given circuit, we assume those subcircuits to be minimal, and we denote by $C[v]$ the (minimall) subcircuit for v in C.

Definition 4. 2: Let C be an arithmetic circuit and let D and E be two disjoint subcircuits of C. A *bridge between D and E* is an edge connecting a gate of D with one of E. We denote by $Bridge(D, E)$ the set of all bridges between D and E. For a gate v of C, we denote by v_{left} and v_{right}, respectively, the gates for the left and right inputs of v. Then we say that a gate v is *weakly skew* if either the gate is skew or $\min\{\#Bridge(C-C[v_{\text{left}}], C[v_{\text{left}}]), \#Bridge(C-C[v_{\text{right}}], C[v_{\text{right}}])\}=1$ (i.e., either $Bridge(C-C[v_{\text{left}}], C[v_{\text{left}}]) = \{(v_{\text{left}}, v)\}$ or $Bridge(C-C[v_{\text{right}}], C[v_{\text{right}}]) = \{(v_{\text{right}}, v)\}$). We say C is *weakly skew* if all multiplication gates of C are weakly skew. We define WSkewARC-SIZE(poly) as the class of all families of polynomials computed by families of polynomial size bounded weakly skew arithmetic circuits.

Intuitively speaking, an arithmetic circuit is weakly skew if for every multiplication gate and at least one of two subcircuits for the inputs of the gate, there is no connection line between the subcircuit and other gates in the remaining part of the circuit, that is, the subcircuit contributes only to the gate. This is a similarity between weakly skew arithmetic circuits and arithmetic formulas, since we have the case for all gates in any arithmetic formulas. By this intuition, it is clear that an arithmetic formula is a weakly skew arithmetic circuit. Furthermore, a skew arithmetic circuit is weakly skew, by the definition.

From these observations and Theorem 3. 2, we easily see that ARF-SIZE(poly) $\subseteq$ DET(poly) = SkewARC-SIZE(poly) $\subseteq$ WSkewARC-SIZE(poly). We below show that the last class actually equals DET(poly).

Theorem 4. 3: For all weakly skew arithmetic circuits C of size q, there exists a square matrix M of dimension at most $2q+2$ such that $P(C)=\det(M)$. Thus, DET(poly) = WSkewARC-SIZE(poly). As an additional observation, when we compute the determinant of a matrix of dimension q by a weakly skew arithmetic circuit, we can reduce the size of the circuit to $\mathcal{O}(q^7)$ from (q^{20}) in Lemma 3. 4.

Proof: The proof is similar to that of Lemma 3. 5. Therefore, we give an outline of the proof to the reader. Let C be a weakly skew arithmetic circuit of size q. Below, we show that we can construct a digraph of at most $2q+2$ nodes that has the same property as G_1 in Lemma 3. 5; that is, the digraph satisfies that for two specified nodes s and t, $P(C)$ is given by $\Sigma_p\{$product of all weights on $p\}$ where the summation is taken over all (s, t)-paths in the digraph.

The construction is done by induction on the number of non-skew multiplication gates. If all multiplication gates are skew, then we can do that as in Lemma 3. 5. For an induction step, assume that for some $k>0$, if C contains less than k non-skew multiplication gates, then we can construct a desired digraph. Consider the case that C has k non-skew multiplication gates and take a non-skew multiplication gate v. We can assume, without loosing the generality, that the subcircuit for the left input of v is connected to the remaining subcircuit with only one bridge (i.e., $Bridge(C-C[v_{\text{left}}], C[v_{\text{left}}])$ contains only one edge (v_{left}, v)). We define two circuits C_1 and C_2 as follows: C_1 is obtained from C by replacing the subcircuit $C[v_{\text{left}}]$ by a new indeterminate y, and $C_2 = C[v_{\text{left}}]$. Then, for each $i=1, 2$, we can construct, by induction hypothesis, a digraph G_i with two specified node s_i and t_i such that $P(C_i)$ is given by $\Sigma_p\{$product of all weights on $p\}$ where the summation is taken over all (s, t)-paths in G_i. Define G to be a digraph obtained by the following consrruction.

(1) For all edges (w, v) in G_1 going into v, we introduce the edge (w, s_2) of the same weight as (w, v).

(2) For all edges (v, w) in G_2 coming out from v, we introduce the edge (t_2, w) of weight one (Note that the edge (v, w) is of weight y in G_1).

(3) We delete v and all edges incident with the node from the digraph obtained above.

Then it is easy to see that $P(C)$ is obtained as $\Sigma_p\{\text{product of all weights on } p\}$ where the summation is taken over all (s_1, t_1)-paths in G. Let q_1 and q_2 denote the number of gates in C_1 and C_2, respectively. By induction hypothesis, each G_i contains at most $2q_i+2$ nodes. On the other hand, since the node representing y in C_1 is eventually deleted in the construction of G_1, we see that G_1 has at most $2q_1+1$ nodes. Hence, G has at most $(2q_1+1)+2q_2+2-1=2q+2$ nodes.

To obtain the latter observation, we may provide some separated skew arithmetic circuits for computing each $C[t]_{i,j}$ in Step 1 and for computing the product in Step 2 of the algorithm described in Lemma 3.4, and after that, we may introduce some required connection lines among those circuits. By such a construction, we can make a weakly skew arithmetic circuit of size $\mathcal{O}(q^7)$ for performing the algorithm. The detail is left to the reader. □

Definition 4.4: Let $\mathcal{P}=\{P_n\}$, $n\geqq 1$ be a family of multivariate polynomials. Below we shall refer each P_n to as its name rather than the polynomial itself. Then we inductively define a class of *arithmetic formulas relative to* $\mathcal{P}$ (*$\mathcal{P}$-formulas* for short), and define the size of those formulas, as follows:

(1) An arithmetic formulas (over $\mathcal{F}$) is a $\mathcal{P}$-formula of same size.

(2) For a P_n and n $\mathcal{P}$-formulas $F^{(1)}, \cdots, F^{(n)}$, $P_n(F^{(1)}, \cdots, F^{(n)})$ is a $\mathcal{P}$-formula of size $1+\Sigma_i size(F^{(i)})$, where $size(F^{(i)})$ denotes the size of $F^{(i)}$. This formula is defined to compute the polynomial $P_n(P(F^{(1)}), \cdots, P(F^{(n)}))$.

(3) For two $\mathcal{P}$-formulas F and G, $(F+G)$ and $F\times G$ are $\mathcal{P}$-formulas of size $1+size(F)+size(G)$, respectively.

(4) What is constructed above is all of the $\mathcal{P}$-formulas.

We say that a family $\mathcal{Q}=\{Q_n\}$, $n\geqq 1$ of polynomials is *polynomial size ARF-reducible* to $\mathcal{P}$ ($\mathcal{Q}\leqq^{ARF}\mathcal{P}$) if there exists a family $\{F_n\}$, $n\geqq 1$ of polynomial size bounded $\mathcal{P}$-formulas such that for all n, $Q_n=P(F_n)$. For a class $\mathbf{C}$ of families of polynomials, we denote by $\leqq^{ARF}(\mathbf{C})$ the class of all families of polynomials that are $\leqq^{ARF}$-reducible to families in $\mathbf{C}$. We say $\mathbf{C}$ is closed downward under the $\leqq^{ARF}$-reducibility if $\leqq^{ARF}(\mathbf{C})=\mathbf{C}$.

Corollary 4.5: $\leqq^{ARF}(\mathrm{DET}(\mathrm{poly}))=\mathrm{DET}(\mathrm{poly})$. Thus DET(poly) is closed downward under the $\leqq^{ARF}$-reducibility.

Proof: It follows from Theorem 3.2 that $\leqq^{ARF}(\mathrm{DET}(\mathrm{poly}))=\leqq^{ARF}(\mathrm{SkewARC\text{-}SIZE}(\mathrm{poly}))$. Then we only note that $\leqq^{ARF}(\mathrm{SkewARC\text{-}SIZE}(\mathrm{poly}))$ is an alternative definition of WSkewARC-SIZE(poly). Thus the corollary follows. □

In the complexity theory of parallel computations based on Boolean circuits, the NC^1-reducibility is usually used in order to classify the computational problems with their parallel complexity. We briefly define an algebraic analogue to the reducibility notion. In what follows, we use the notations in Definition 4.4.

Definition 4.6: By an *oracle gate for* $\mathcal{P}$ we mean a single gate that computes some P_n in $\mathcal{P}$, and by the *depth* of a given circuit possibly including some oracle gates, we mean the length of the longest path from an input gate to the output gate of the circuits, where an oracle gate with m inputs contributes $[\log_2 m]$ to the depth. We say that the family $\mathcal{Q}$ is $\leqq^{\mathrm{NC}^1\mathcal{F}}$-*reducible* to $\mathcal{P}$ ($\mathcal{Q}\leqq^{\mathrm{NC}^1\mathcal{F}}\mathcal{P}$) if there exists a family of $\mathcal{O}(\log_2 n)$ depth bounded arithmetic circuits with oracle gates for $\mathcal{P}$ that computes $\mathcal{Q}$. Note that the size of d depth bounded arithmetic circuits with any oracle gates is bounded above by 2^d where an oracle gate contributes the number of its inputs to the size; hence, all circuits in the above family is of size polynomial in n. We denote by $\leqq^{\mathrm{NC}^1\mathcal{F}}(\mathbf{C})$ the class of families of polynomials $\leqq^{\mathrm{NC}^1\mathcal{F}}$-reducible to families in $\mathbf{C}$.

Corollary 4.7: $\leqq^{\mathrm{NC}^1\mathcal{F}}(\mathrm{DET}(\mathrm{poly}))=\mathrm{DET}(\mathrm{poly})$. Thus DET(poly) is closed downward under the $\leqq^{\mathrm{NC}^1\mathcal{F}}$-reducibility.

Proof: Given an arithmetic circuit C of depth d with any oracle gates for some family $\mathcal{P}$, we can obtain the *tree-equivalent* circuit T_C with the same depth, by replicating all gates with fan-out more than one, which computes the same polynomial as C does. This implies that $P(C)$ is computed by a $\mathcal{P}$-formula, i.e., T_C, of size at most 2^d. The corollary follows from this last observation. The detail is left to the reader. □

Closing this section, we demonstrate a difference between ARF-SIZE(poly) and DET(poly). By a non-weakly-skew (multiplication and addition) gate, we mean a (multiplication and addition, respectively) gate that is not weakly skew. We can look at a difference on the number of non-weakly-skew addition gates used in polynomial size bounded arithmetic circuits for computing the families of polynomials in both classes.

Theorem 4.8

(1) A family of polynomials is in DET(poly) iff it can be computed by a family of polynomial size bounded arithmetic circuits with $\mathcal{O}(\log_2 n)$ non-weakly-skew multiplication gates (and with a polynomial number of non-weakly-skew addition gates).

(2) A family of polynomials is in ARF-SIZE(poly) iff it can be computed by a family of polynomial size bounded arithmetic circuits with $\mathcal{O}(\log_2 n)$ non-weakly-skew (addition and multiplication) gates.

Proof: Given an arithmetic circuit of size s that contains m non-weakly-skew (resp., multiplication) gates, we can obtain an arithmetic formula (sesp., a

weakly skew arithmetic circuit) of size at most $s+s \cdot 2^m$ by repeating the following process until the resulting circuit has no non-weakly-skew (resp., multiplication) gate: for each non-weakly-skew (resp., multiplication) gate, add to the circuit a copy of the subcircuit for an input of the gate, and use the copy, instead of the original one, for the input of the gate. It is easy to see that the resulting circuit has the size bound above. The theorem follows immediately from this observation. □

Note that an arithmetic circuit of bounded degree has much more freedom than weakly skew ones on the usage of multiplication gates, while all of the freedom is not allowed to it. Thus we may intuitively say that in a concise computation for the determinant, addition gates may be used as in the general circuits (and hence as in the degree bounded case) but multiplication gates must be used as in arithmetic formulas.

5. Concluding Remarks

In this paper, we have defined the class of skew arithmetic circuits whose size is polynomially related to the dimension of square matrices when we compute the determinant. In addition, we have extended the result to other classes such as weakly skew circuits. This work has been motivated by the question of when the polynomials defined as the determinant differ from those polynomials computed by arithmetic formulas and by arithmetic circuits of relatively small size and degree. From our results, we intuitively saw that arithmetic circuits for computing the determinant use multiplication gates just as in the arithmetic formulas and use addition gates just as in the general arithmetic circuits (hence, as in the degree bounded case).

The above differences based on our results may be said to indicate *qualitative or structural* differences on the three classes. On the other hand, we are interested in finding some *quantitative* differences among the classes by comparing the amount of some resources in a computational model for performing the computations in those classes. Circuit size, depth, and degree do not seem to work well for this purpose. At the end of the last section, we have shown a result along this line, but we do not find satisfaction in the result. It seems necessary for this purpose to find some good complexity measure on a (possibly new) computational model for evaluating the formal polynomial over a field. We believe that such a further investigation is meaningful for making the nature of the determinant clear from the complexity theoretic viewpoint.

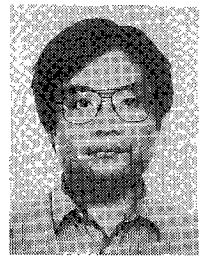
It would also be interesting to find a different class of arithmetic circuits that can compute the determinant more effciently than weakly skew circuits and can be simulated by the determinant as efficiently as weakly skew arithmetic circuits. The best hope is to find a class of circuits whose size is linearly related to the dimension of matrices when we consider the simulation of those circuits by the determinant and vice versa (in this sense, the size of all classes in the present paper is polynomially related to the dimension of matrices).

Acknowledgement

I am very thankful to the anonymous referees for their improvements on the presentation of this paper and their corrections on errors in the earlier version of this paper.

References

(1) Berkowitz S. J.: "On Computing the Determinant in Small Parallel Time Using a Small Number of Processors", Inform. Process. Letters, **18**, pp. 147–150 (1984).
(2) Cook S. A.: "A Taxonomy of Problems with Fast Parallel Algorithms", Information and Control, **64**, pp. 2–22 (1985).
(3) Valiant L. G.: "Completeness Classes in Algebra", Proc. 11th ACM Symposium on the Theory of Computing, pp. 249–261 (1979).
(4) Valiant L. G., Skyum S., Berkowitz N. and Rackoff C.: "Fast Parallel Computation of Polynomials Using Few Processors", SIAM J. on Computing, **12**, pp. 641–644 (1983).
(5) Venkateswaran H.: "Circuit Definition of Nondeterministic Complexity Classes", Proc. the 8th Conference on Foundations of Software Technology and Theoretical Computer Science, Lecture Notes in Computer Sciences, **338**, pp. 175–192, Springer-Verlag (1988).



Seinosuke Toda was born in Hyogo, Japan, on January 15, 1959. He received M. E. degree in Computer Science from University of Electro-Communications in 1984 and received D. S. degree from the Tokyo Institute of Technology in 1991. In 1984 he joined National Institute of Japanese Literatures as a research associate, and since 1988, he is a research associate of the Department of Computer Science and Information Mathematics of University of Electro-Communications. He is mainly interested in drawing the map of complexity classes and analyzing the complexity of concrete computational problems.