REDUCING RANDOMNESS VIA IRRATIONAL NUMBERS[*]

ZHI-ZHONG CHEN[†] AND MING-YANG KAO[‡]

Abstract. We propose a general methodology for testing whether a given polynomial with integer coefficients is identically zero. The methodology evaluates the polynomial at efficiently computable approximations of suitable irrational points. In contrast to the classical technique of DeMillo, Lipton, Schwartz, and Zippel, this methodology can decrease the error probability by increasing the precision of the approximations instead of using more random bits. Consequently, randomized algorithms that use the classical technique can generally be improved using the new methodology. To demonstrate the methodology, we discuss two nontrivial applications. The first is to decide whether a graph has a perfect matching in parallel. Our new NC algorithm uses fewer random bits while doing less work than the previously best NC algorithm by Chari, Rohatgi, and Srinivasan. The second application is to test the equality of two multisets of integers. Our new algorithm improves upon the previously best algorithms by Blum and Kannan and can speed up their checking algorithm for sorting programs on a large range of inputs.

Key words. polynomial identification, Galois theory, randomized algorithms, parallel algorithms, program checking, perfect matchings, multiset equality test

AMS subject classifications. 12F10, 68Q25, 68Q22, 65Y05, 68R10, 05C70

PII. S0097539798341600

1. Introduction. Many algorithms involve checking whether certain polynomials with integer coefficients are identically zero. Oftentimes, these polynomials have exponential-sized standard representations while having succinct nonstandard representations [6, 17, 18, 22]. This paper focuses on testing such polynomials with integer coefficients.

Given a polynomial $Q(x_1, \ldots, x_q)$ in a succinct form, a naive method of testing it is to transform it into the standard simplified form and then test whether its coefficients are all zero. Since Q may have exponentially many monomials, this method may take exponential time. Let d_Q be the degree of Q. DeMillo and Lipton [6], Schwartz [18], and Zippel [22] proposed an advanced method, which we call *the DLSZ method.* It evaluates $Q(i_1, \ldots, i_q)$, where $i_1, \ldots, i_q$ are uniformly and independently chosen at random from a set S of $2d_Q$ integers. This method uses $q\lceil \log(2d_Q)\rceil$ random bits and has an error probability of at most $\frac{1}{2}$. (Every log in this paper is to base 2.) There are three general techniques that use additional random bits to lower the error probability to $\frac{1}{t}$ for any integer $t > 2$. These techniques have their own advantages and disadvantages in terms of the running time and the number of random bits used. The first performs $\lceil \log t\rceil$ independent evaluations of Q at $\lceil \log(2d_Q)\rceil$-bit integers, using $q\lceil \log(2d_Q)\rceil\lceil \log t\rceil$ random bits. The second enlarges the cardinality of S from $2d_Q$ to td_Q and performs one evaluation of Q at $\lceil \log(td_Q)\rceil$-bit integers, using $q\lceil \log d_Q + \log t\rceil$ random bits. The third is *probability amplification* [15]. A basic such technique works for $t \leq 2^{q\lceil \log(2d_Q)\rceil}$ by performing t pairwise independent evaluations of Q at

$\lceil \log(2d_Q) \rceil$-bit integers, using $2q\lceil \log(2d_Q) \rceil$ random bits. Stronger amplification can be obtained by means of random walks on expanders [5, 1, 8].

In section 2, we propose a new general methodology for testing $Q(x_1, \ldots, x_q)$. Our methodology computes $Q(\pi_1, \ldots, \pi_q)$, where $\pi_1, \ldots, \pi_q$ are suitable irrational numbers such that $Q(\pi_1, \ldots, \pi_q) = 0$ if and only if $Q(x_1, \ldots, x_q) \equiv 0$. Since rational arithmetic is used in actual computers, we replace each π_i with a rational approximation π_i'. A crucial question is how many bits each π_i' needs to ensure that $Q(\pi_1', \ldots, \pi_q') = 0$ if and only if $Q(x_1, \ldots, x_q) \equiv 0$. We give an explicit answer to this question, from which we obtain a new randomized algorithm for testing Q. Our algorithm runs in polynomial time and uses $\sum_{i=1}^q \lceil \log(d_i + 1) \rceil$ random bits, where d_i is the degree of x_i in Q. Moreover, the error probability can be made inverse polynomially small by increasing the bit length of each π_i'. Thus, our methodology has two main advantages over previous techniques:

- It uses fewer random bits if some d_i is less than d_Q.
- It can reduce the error probability without using one additional random bit.

In general, randomized algorithms that use the classical DeMillo, Lipton, Schwartz, and Zippel (DLSZ) method can be improved using the new methodology. To demonstrate the methodology, we discuss two nontrivial applications. In section 3, the first application is to decide whether a given graph has a perfect matching. This problem has deterministic polynomial-time sequential algorithms but is not known to have a deterministic NC algorithm [7, 10, 13, 21]. We focus on solving it in parallel using as few random bits as possible. Our new NC algorithm uses fewer random bits while doing less work than the previously best NC algorithm by Chari, Rohatgi, and Srinivasan [4]. In section 4, the second application is to test the equality of two given multisets of integers. This problem was initiated by Blum and Kannan [3] for checking the correctness of sorting programs. Our new algorithm improves upon the previously best algorithms developed by them and can speed up their checking algorithm for sorting programs on a large range of inputs.

2. A new general methodology for testing polynomials. The following notation is used throughout this paper.

- Let $Q(x_1, \ldots, x_q)$ be a polynomial with integer coefficients; we wish to test whether $Q(x_1, \ldots, x_q) \equiv 0$.
- For each x_i, let d_i be an upper bound on the degree of x_i in Q. Let $k_i = \lceil \log(d_i + 1) \rceil$.
- Let $k = \max_{i=1}^q k_i$ and $K = \sum_{i=1}^q k_i$; K is the number of random bits used by the methodology as shown in Theorem 2.3.
- Let d be an integer upper bound on the degree of Q; without loss of generality, we assume $d \geq \max_{i=1}^q d_i$.
- Let c be an upper bound on the absolute value of a monomial's coefficient in Q.
- Let Z be an upper bound on the number of monomials in Q; without loss of generality, we assume $Z \leq \sum_{i=0}^d q^i$.
- Let $\psi = \log c + \log Z + d(\log k + \frac{\log K}{2} + \log \ln K)$. Let ℓ be an integer at least $\psi + 1 + \log d$; ℓ determines the precision of our approximation to the irrational numbers chosen for the variables x_i.

For example, if all $d_i = 1$, then $k_i = 1$, $K = q$, and our goal is to use exactly q random bits, i.e., one bit per variable x_i.

LEMMA 2.1. *Let $p_{1,1}, \ldots, p_{1,k_1}, \ldots, p_{q,1}, \ldots, p_{q,k_q}$ be K distinct primes. For each $p_{i,j}$, let $b_{i,j}$ be a bit. For each x_i, let $\pi_i = \sum_{j=1}^{k_i} (-1)^{b_{i,j}} \sqrt{p_{i,j}}$. Then $Q(x_1, \ldots, x_q) \not\equiv 0$*

if and only if $Q(\pi_1, \ldots, \pi_q) \neq 0$.

Proof. This lemma follows from Galois theory in algebra [14]. Let $A_0 = B_0$ be the field of rational numbers. For each x_j, let $K_j = \sum_{i=1}^{j} k_i$. Let A_j be the field generated by $\pi_1, \pi_2, \ldots, \pi_j$ over A_0. Let B_j be the field generated by $p_{1,1}, \ldots, p_{1,k_1}, \ldots, p_{j,1}, \ldots, p_{j,k_j}$ over B_0. By induction, $A_j = B_j$, the dimension of A_j over A_0 is 2^{K_j}, and the dimension of A_j over A_{j-1} is 2^{k_j}. Thus, π_j is not a root of any nonzero single variate polynomial over A_{j-1} that has a degree less than 2^{k_j}. Since $d_j < 2^{k_j}$, by induction, $Q(\pi_1, \ldots, \pi_j, x_{j+1}, \ldots, x_q) \not\equiv 0$. The lemma is proved at $j = q$. $\square$

In light of Lemma 2.1, the next algorithm tests $Q(x_1, \ldots, x_q)$ by approximating the irrational numbers $\sqrt{p_{i,j}}$ and randomizing the bits $b_{i,j}$.

ALGORITHM 1.

(1) Compute $q, d_1, \ldots, d_q, k_1, \ldots, k_q, K, d, c, Z$.
(2) Choose $p_{1,1}, \ldots, p_{1,k_1}, \ldots, p_{q,1}, \ldots, p_{q,k_q}$ to be the K smallest primes.
(3) Choose each $b_{i,j}$ independently with equal probability for 0 and 1.
(4) Pick ℓ, which determines the precision of our approximation to $\sqrt{p_{i,j}}$.
(5) For each $p_{i,j}$, compute a rational number $r_{i,j}$ from $\sqrt{p_{i,j}}$ by cutting off the bits after the ℓth bit after the decimal point.
(6) Compute $\Delta = Q(\sum_{j=1}^{k_1}(-1)^{b_{1,j}} r_{1,j}, \ldots, \sum_{j=1}^{k_q}(-1)^{b_{q,j}} r_{q,j})$.
(7) Output "$Q(x_1, \ldots, x_q) \not\equiv 0$" if and only if $\Delta \neq 0$.

The next lemma shows how to choose an appropriate ℓ at Step 1 of Algorithm 1.

LEMMA 2.2. *If $Q(x_1, \ldots, x_q) \not\equiv 0$, then $|\Delta| \geq 2^{-\ell}$ with probability at least $1 - \frac{\psi}{\ell - 1 - \log d}$.*

Proof. For each combination of the bits $b_{i,j}$, $Q(\pi_1, \ldots, \pi_q)$ is called a *conjugate*. By the prime number theorem [11], $\sqrt{p_{i,j}} \leq \sqrt{K} \ln K$ and thus $|\pi_i| \leq k\sqrt{K} \ln K$. Then, since Q has at most Z monomials, each conjugate's absolute value is at most $2^\psi = cZ(k\sqrt{K} \ln K)^d$. Let $\ell' = \ell - \psi - 1 - \log d$. Let α be the number of the conjugates that are less than $2^{-\ell'}$. Let $\beta = 2^K - \alpha$ be the number of the other conjugates. Let Π be the product of all the conjugates. By Lemma 2.1, $\Pi \neq 0$, and by algebra [9], Π is an integer. Thus, $|\Pi| \geq 1$ and $\alpha(-\ell') + \beta\psi \geq 0$. Hence, $\frac{\beta}{2^K} \geq \frac{\ell'}{\ell' + \psi}$; i.e., $|Q(\pi_1, \ldots, \pi_q))| \geq 2^{-\ell'}$ with the desired probability. We next show that if $|Q(\pi_1, \ldots, \pi_q)| \geq 2^{-\ell'}$, then $|\Delta| \geq 2^{-\ell}$. Since $r_{i,j} > \sqrt{p_{i,j}} - 2^{-\ell}$, $\sum_{j=1}^{k_i} r_{i,j} > |\pi_i| - k2^{-\ell}$. So approximating $p_{i,j}$ reduces each monomial term's absolute value in $Q(\pi_1, \ldots, \pi_q)$ by at most $c(k\sqrt{K} \ln K)^{d-1} dk2^{-\ell}$. Thus, $|\Delta| \geq |Q(\pi_1, \ldots, \pi_q)| - cZ(k\sqrt{K} \ln K)^d 2^{-\ell + \log d} \geq |Q(\pi_1, \ldots, \pi_q)| - 2^{-\ell' - 1} \geq 2^{-\ell}$. $\square$

THEOREM 2.3. *For a given $t > 1$, set $\ell \geq t\psi + 1 + \log d$. If $Q(x_1, \ldots, x_m) \equiv 0$, Algorithm 1 always outputs the correct answer; otherwise, it outputs the correct answer with probability at least $1 - \frac{1}{t}$. Moreover, it uses exactly K random bits, and its error probability can be decreased by increasing t without using one additional random bit.*

Proof. This theorem follows from Lemma 2.2 immediately. $\square$

Let $||Q||$ be the size of the input representation of Q. The next lemma supplements Theorem 2.3 by discussing sufficient conditions for Algorithm 1 to be efficient.

LEMMA 2.4. *With $Z = \sum_{i=1}^{d} q^i$, Algorithm 1 takes polynomial time in $||Q||$ and t under the following conditions:*

- *The parameters $q, d_1, \ldots, d_q, d$ are at most $(t||Q||)^{O(1)}$ and are computable in time polynomial in $t||Q||$.*
- *The parameter c is at most $2^{O(t||Q||)}$ and is computable in time polynomial in $t||Q||$.*

- *Given ℓ'-bit numbers p_i', $Q(p_1', \ldots, p_q')$ is computable in time polynomial in $t\|Q\|$ and ℓ'.*

Proof. The proof is straightforward based on the following key facts. There are at most $(t\|Q\|)^{O(1)}$ primes $p_{i,j}$, which can be efficiently found via the prime number theorem. Each $r_{i,j}$ has at most $(t\|Q\|)^{O(1)}$ bits and can be efficiently computed by, say, Newton's method. $\square$

We can scale up the rationals $r_{i,j}$ to integers and then compute Δ modulo a reasonably small random integer. As shown in later sections, this may considerably improve the efficiency of Algorithm 1 by means of the next fact.

FACT 1 (Thrash [19]). *Let $h \geq 3$ be an integer. If H is a subset of $\{1, 2, \ldots, h^2\}$ with $|H| \geq \frac{h^2}{2}$, then the least common multiple of the elements in H exceeds 2^h. Thus, for a given positive integer $h' \leq 2^h$, a random integer from $\{1, 2, \ldots, h^2\}$ does not divide h' with probability at least $\frac{1}{2}$.*

3. Application to perfect matching test. Let $G = (V, E)$ be a graph with n vertices and m edges. Let $V = \{1, 2, \ldots, n\}$. Without loss of generality, we assume that n is even and $m \geq \frac{n}{2}$. A *perfect matching* of G is a set L of edges in G such that no two edges in L have a common endpoint and every vertex of G is incident to an edge in L.

Given G, we wish to decide whether it has a perfect matching. This problem is not known to have a deterministic NC algorithm. The algorithm of Chari, Rohatgi, and Srinivasan [4] uses the fewest random bits among the previous NC algorithms. This paper gives a new algorithm that uses fewer random bits while doing less work. For ease of discussion, a detailed comparison is made right after Theorem 3.2.

3.1. Classical ideas. The *Tutte matrix* of G is the following $n \times n$ skew-symmetric matrix M of m distinct indeterminates $y_{i,j}$:

$$M_{i,j} = \left\{ \begin{array}{rl} y_{i,j} & \text{if } \{i,j\} \in E \text{ and } i < j, \\ -y_{j,i} & \text{if } \{i,j\} \in E \text{ and } i > j, \\ 0 & \text{otherwise.} \end{array} \right.$$

Let $L = \{\{i_1, j_1\}, \ldots, \{i_{\frac{n}{2}}, j_{\frac{n}{2}}\}\}$ be a perfect matching of G, where $i_1 < j_1, i_2 < j_2, \ldots, i_{\frac{n}{2}} < j_{\frac{n}{2}}$ and $i_1 < i_2 < \cdots < i_{\frac{n}{2}}$. Let $\pi(L) = y_{i_1,j_1} y_{i_2,j_2} \cdots y_{i_{\frac{n}{2}},j_{\frac{n}{2}}}$. Let $\sigma(L) = 1$ or -1 if the following permutation is even or odd, respectively:

$$\left(\begin{array}{ccccc} 1 & 2 & \cdots & n-1 & n \\ i_1 & j_1 & \cdots & i_{\frac{n}{2}} & j_{\frac{n}{2}} \end{array} \right).$$

Let $\mathrm{Pf}(G) = \sum_L \pi(L)\sigma(L)$, where L ranges over all perfect matchings in G.

FACT 2 (Fisher and Kasteleyn [2], Tutte [20]).
- $\det M = (\mathrm{Pf}(G))^2$.
- *G has a perfect matching if and only if $\det M \not\equiv 0$.*

Combining Fact 2 and the DLSZ method, Lovasz [12] gave a randomized NC algorithm for the matching problem. Since the degree of $\det M$ is at most n, this algorithm assigns to each $x_{i,j}$ a random integer from $\{1, 2, \ldots, 2n\}$ uniformly and independently and outputs "G has a perfect matching" if and only if $\det M$ is nonzero at the chosen integers. Its error probability is at most $\frac{1}{2}$, using $m\lceil \log(2n) \rceil$ random bits. The time and processor complexities are dominated by those of computing the determinant of an $n \times n$ matrix with $O(\log n)$-bit integer entries.

3.2. A new randomized NC algorithm. A direct application of Theorem 2.3 to det M uses $O(m)$ random bits, but our goal is $O(n + \log m/n)$ bits. Therefore, we need to reduce the number of variables in det M.

- Let G' be the acyclic digraph obtained from G by orienting each edge $\{i, j\}$ into the arc $(\min\{i,j\}, \max\{i,j\})$.
- For each vertex i in G', let n_i be the number of outgoing arcs from i.
- Let $\hat{n}_i = 0$ if $n_i = 0$; otherwise, $\hat{n}_i = \lceil \log n_i \rceil$.
- Let $q = \sum_{i=1}^{n} \hat{n}_i$. Note that $q < n + n \log \frac{m}{n}$.
- Let $x_1, x_2, \ldots, x_q$ be q distinct new indeterminates.

We label the outgoing arcs of each vertex as follows. If $n_1 = 0$, vertex 1 has no outgoing arc in G'. If $n_1 = 1$, label its unique outgoing arc with 1. If $n_1 \geq 2$, label its n_1 outgoing arcs each with a distinct monomial in $\{(x_1)^{a_1}(x_2)^{a_2} \cdots (x_{\hat{n}_1})^{a_{\hat{n}_1}} \mid \text{each } a_h \text{ is } 0 \text{ or } 1\}$, which is always possible since $2^{\hat{n}_1} \geq n_1$. We label the n_2 outgoing arcs of vertex 2 in the same manner using $x_{\hat{n}_1+1}, x_{\hat{n}_1+2}, \ldots, x_{\hat{n}_1+\hat{n}_2}$. We similarly process the other vertices i, each using the next $\hat{n}_i$ available indeterminates x_h.

Let $f_{i,j}$ be the label of arc (i,j) in G'. Let $Q(x_1, \ldots, x_q)$ be the polynomial obtained from $\mathrm{Pf}(G)$ by replacing each indeterminate $y_{i,j}$ with $f_{i,j}$.

LEMMA 3.1. *G has a perfect matching if and only if $Q(x_1, \ldots, x_q) \not\equiv 0$.*

Proof. For each L as described in section 3.1, let $Q_L = \sigma(L) f_{i_1,j_1} f_{i_2,j_2} \cdots f_{i_{\frac{n}{2}},j_{\frac{n}{2}}}$. Then $Q = \sum_L Q_L$, where L ranges over all the perfect matchings of G. It suffices to prove that for distinct perfect matchings L_1 and L_2, the monomials Q_{L_1} and Q_{L_2} differ by at least one x_h. Let H be the subgraph of G induced by $(L_1 \cup L_2) - (L_1 \cap L_2)$. H is a set of vertex-disjoint cycles. Since $L_1 \neq L_2$, H contains at least one cycle C. Let C' be the acyclic digraph obtained from C by replacing each edge $\{i,j\}$ with the arc $(\min\{i,j\}, \max\{i,j\})$. C' contains two outgoing arcs (i, j_1) and (i, j_2) of some vertex i. So there is an indeterminate x_h used in arc labels for vertex i, whose degree is 1 in one of f_{i,j_1} and f_{i,j_2} but is 0 in the other. Hence, the degree of x_h is 1 in one of Q_{L_1} and Q_{L_2} but is 0 in the other, which makes Q_{L_1} and Q_{L_2} distinct as desired. □

To test whether G has a perfect matching, we use Algorithm 1 to test Q by means of Theorem 2.3 and Lemma 3.1. Below we detail each step of Algorithm 1.

Step 1. Compute q. Then set $d_1 = d_2 = \cdots = d_q = 1$, $k_1 = k_2 = \cdots = k_q = 1$, $K = q$, $d = q$, $c = 1$. Further set $Z = (\frac{2m}{n})^n$ since the number of perfect matchings in G is at most $\Pi_{i=1}^n m_i \leq (\frac{2m}{n})^n$, where m_i is the degree of node i in G.

Step 2. This step computes the q smallest primes $p_{1,1}, p_{2,1}, \ldots, p_{q,1}$, each at most $q \ln^2 q$. Since a positive integer p is prime if and only if it is indivisible by any integer i with $2 \leq i \leq \sqrt{p}$, these primes can be found in $O(\log q)$ parallel arithmetic steps on integers of at most $\lceil \log(1 + q \ln^2 q) \rceil$ bits using $O(q^{1.5} \log^3 q)$ processors.

Step 3. This step is straightforward.

Step 4. Set $\ell = \lceil t\psi \rceil + \lceil q \rceil + 1$, where $\psi = n \log \frac{2m}{n} + q \log(\sqrt{q} \ln q)$.

Step 5. We use Newton's method to compute $r_{i,1}$ from $p_{i,1}$. For the convenience of the reader, we briefly sketch the method here. We use $g_0 = p_{i,1}$ as the initial estimate. After the jth estimate g_j is obtained, we compute $g_{j+1} = \frac{1}{2}(g_j + \frac{p_{i,1}}{g_j})$, maintaining only the bits of g_{j+1} before the $(\ell + 1)$th bit after the decimal point. Thus, $g_{j+1} \leq \frac{1}{2}(g_j + \frac{p_{i,1}}{g_j})$. With g_{j+1} obtained, we check whether $g_{j+1}^2 > p_{i,1}$. If not, we stop; otherwise, we proceed to compute g_{j+2}. Since the convergence order of the method is 2, we take the $\lceil \log(\lceil \log p_{i,1} \rceil + \ell) \rceil$th estimate as $r_{i,1}$. Thus $r_{1,1}, \ldots, r_{q,1}$ can be computed in $O(\log(\ell + \log q))$ parallel arithmetic steps with q processors. Note that each g_j has at most $\lceil \log(1 + q \ln^2 q) \rceil + \ell$ bits.

Step 6. Evaluating Δ is equivalent to computing Δ^2. Δ^2 is the determinant of an $n \times n$ skew-symmetric matrix M' whose nonzero entries above the main diagonal in the ith row are either 1 or products of at most $\hat{n}_i$ rationals among $r_{1,1}, \ldots, r_{q,1}$. Thus, each matrix entry has at most $\lceil \log n \rceil (\lceil \log(1 + q \ln^2 q) \rceil + \ell)$ bits. Setting up M' takes $O(\log n)$ arithemetic steps on $O(n^2)$ processors.

Step 7. This step is straightforward.

The next theorem summarizes the above discussion.

THEOREM 3.2. *For any given $t > 1$, whether G has a perfect matching can be determined in $O(\log(nt))$ parallel arithmetic steps on rationals of $O(tn \log^3 n)$ bits using $O(n^2)$ processors together with one evaluation of the determinant of an $n \times n$ matrix of $O(tn \log^3 n)$-bit rational entries. The error probability is at most $\frac{1}{t}$, using $q < n + n \log \frac{m}{n}$ random bits.*

Remark. The best known NC algorithm for computing the determinant of an $n \times n$ matrix takes $O(\log^2 n)$ parallel arithmetic steps using $O(n^{2.376})$ processors [16].

Proof. We separate the total complexity of Algorithm 1 into that for computing $\det M'$ and that for all the other computation. For the latter, the running time is dominated by that of Step 5; the bit length by that of the entries in M' at Step 6; and the processor count by that of setting up M'. $\square$

The work of Chari, Rohatgi, and Srinivasan [4] aims to use few random bits when the number of perfect matchings is small. Indeed, their algorithm uses the fewest random bits among the previous NC algorithms. For an error probability at most $\frac{3}{4}$, it uses $\min\{28 \sum_{i=1}^n \lceil \log \hat{d}_i \rceil, 6m + 4 \sum_{i=1}^n \lceil \log \hat{d}_i \rceil\} + O(\log n)$ random bits, where $\hat{d}_i$ is the degree of vertex i in G. It also computes the determinant of an $n \times n$ matrix with $O(n^7)$-bit entries. In contrast, with $t = 2$ in Theorem 3.2, Algorithm 1 has an error probability at most $\frac{1}{2}$ while using fewer random bits, i.e., $q < n + n \log \frac{m}{n}$ bits. Moreover, using the best known NC algorithm for determinants, the work of Algorithm 1 is dominated by that of computing the determinant of an $n \times n$ matrix with entries of shorter length, i.e., $O(n \log^3 n)$ bits.

The next theorem modifies the above implementation of Algorithm 1 by means of Fact 1 so that it computes the determinants of matrices with only $O(\log(nt))$-bit integer entries but uses slightly more random bits.

THEOREM 3.3. *For any given $t > 2$, whether G has a perfect matching can be determined in $O(\log(nt))$ parallel arithmetic steps on rationals of $O(tn \log^3 n)$ bits using $O(n^2)$ processors together with $\lceil \log t \rceil$ evaluations of the determinant of an $n \times n$ matrix of $O(\log(nt))$-bit integer entries. The error probability is at most $\frac{2}{t}$, using $q + O(\log t \log(nt))$ random bits, which is at most $n + n \log \frac{m}{n} + O(\log t \log(nt))$.*

Proof. We modify Steps 6 and 7 of the above implementation as follows.

Step 6.
- Compute M' as above.
- For each (i,j)th entry of M', we multiply it with $2^{(\hat{n}_i + \hat{n}_j)\ell}$ in $O(1)$ parallel arithmetic steps using $O(n^2)$ processors. Let M'' be the resulting matrix; note that $\det M'' = 2^{2q\ell} \det M'$ and each entry of M'' is an integer of at most $3\lceil \log n \rceil(\ell + \lceil \log n \rceil)$ bits.
- Let $\lambda = \lceil \log t \rceil$. Let $u = n! \cdot 2^{3n \lceil \log n \rceil(\ell + \lceil \log n \rceil)}$; note that $|\det M''| \le u$. We uniformly and independently choose λ random positive integers $w \le \lceil \log u \rceil^2$ using $O(\lambda \log(nt))$ random bits in $O(\lambda)$ steps on a single processor. For each chosen w, we first compute $M''' = M'' \bmod w$ in $O(1)$ parallel arithmetic steps using $O(n^2)$ processors; and then compute $\det M'''$ instead of $\det M'$.

Step 7. Output "G has a perfect matching" if and only if some $\det M'''$ is nonzero.

By Fact 1, if $\det M'' \neq 0$, then some chosen w does not divide $\det M''$ with probability at least $1 - 2^{-\lambda}$. Thus, the overall error probability is at most $\frac{1}{t} + 2^{-\lambda} \leq \frac{2}{t}$. We separate the total complexity of Algorithm 1 into that for computing $\det M'''$ and that for all the other computation. As with Theorem 3.2, the running time of the latter remains dominated by that of Step 5, the bit length by that of the entries in M' at Step 6, and the processor count by that of setting up M'. $\square$

4. Application to multiset equality test. Let $A = \{a_1, \ldots, a_n\}$ and $B = \{b_1, \ldots, b_n\}$ be two multisets of positive integers. Let a be the largest possible value for any element of $A \cup B$. Given A, B, and a as input, the *multiset equality test problem* is that of deciding whether $A \equiv B$, i.e., whether they contain the same number of copies for each element in $A \cup B$. This problem was initiated by Blum and Kannan [3] to study how to check the correctness of sorting programs. They gave two randomized algorithms on a useful model of computation which reflects many sorting scenarios better than the usual RAM model. For brevity, we denote their model by MBK and the two algorithms by ABK_1 and ABK_2.

This section modifies the MBK model to cover a broader range of sorting applications. It then gives a new randomized algorithm, which improves upon ABK_1 and ABK_2 and can speed up the checking algorithm for sorting by Blum and Kannan [3] on a large range of inputs.

4.1. Models of computation and previous results. In both the MBK model and the modified model, the computer has $O(1)$ tapes as well as a random access memory of $O(\log n + \log a)$ words. The allowed elementary operations are $+, -, \times, /, <, =$, and two bit operations shift-to-left and shift-to-right, where $/$ is integer division. Each of these operations takes one step on integers that are one word long; thus the division of an integer of m_1 words by another of m_2 words takes $O(m_1 m_2)$ time. In addition, it takes one step to copy a word on tape to a word in the random access memory or vice versa.

The only difference between the two models is that the modified model has a shorter word length relative to a and therefore is applicable to sorting applications with a larger range of keys. To be precise, in the MBK model, each word has $1 + \lfloor \log a \rfloor$ bits, and thus can hold a nonnegative integer at most a. In the modified model, each word has $\xi = 1 + \lfloor \log \max\{\lceil \log n \rceil, \lceil \log a \rceil\} \rfloor$ bits and thus can hold a nonnegative integer at most $\max\{\lceil \log n \rceil, \lceil \log a \rceil\}$.

Note that sorting A and B by comparison takes $O(n \log n)$ time in the MBK model and $O(\frac{\log a}{\xi} n \log n)$ time in the modified model. However, in both models, if $n \geq 2^a$, the equality of A and B can be tested in optimal $O(n)$ time with bucket sort. Hence, we hereafter assume $n < 2^a$. We briefly review ABK_1 and ABK_2 as follows.

Let $Q_1(x)$ be the polynomial $\sum_{i=1}^{n} x^{a_i} - \sum_{i=1}^{n} x^{b_i}$. ABK_1 selects a random prime $w \leq 3a\lceil \log(n+1) \rceil$ uniformly and computes $Q_1(n+1) \bmod w$ in a straightforward manner. It outputs "$A \equiv B$" if and only if $Q_1(n+1) \bmod w$ is zero. Excluding the cost of computing w, ABK_1 takes $O(n \log a)$ time in the MBK model and $O\left((\frac{\log a}{\xi})^2 n \log a\right)$ time in the modified model. The error probability is at most $\frac{1}{2}$.

Let $Q_2(x)$ be the polynomial $\Pi_{i=1}^{n}(x - a_i) - \Pi_{i=1}^{n}(x - b_i)$. ABK_2 uniformly selects a random positive integer $z \leq 4n$ and a random prime $w \leq 3n\lceil \log(a + 4n) \rceil$; and computes $P(z) \bmod w$ in a straightforward manner. It outputs "$A \equiv B$" if and only if $P(z) \bmod w$ is zero. Excluding the cost of computing w, ABK_2 takes $O\left(n \max\{1, (\frac{\log n}{\log a})^2\}\right)$ time in the MBK model and $O\left(n \frac{(\log n + \log a)(\log n + \log \log a)}{\xi^2}\right)$ time in the modified model. The error probability is at most $\frac{3}{4}$.

Generating the random primes w is a crucial step of ABK_1 and ABK_2. It is unclear how this step can be performed efficiently in terms of running time and random bits. We modify this step by means of Fact 1 as follows. In ABK_1, $|Q_1(n+1)| \leq 2^{1+a\log(n+1)+\log n}$; in ABK_2, $|Q_2(2n)| \leq 2^{1+n\log(a+4n)}$. Thus, we can replace w in ABK_1 and ABK_2 with two random positive integers $w_1 \leq (1 + a\log(n+1) + \log n)^2$ and $w_2 \leq (1 + n\log(a + 4n))^2$, respectively. With these modifications, ABK_1 and ABK_2 use at most $2\log a + 2\log\log n + O(1)$ and $3\log n + 2\log\log(a + n) + O(1)$ random bits, respectively. The time complexities and error probabilities remain as stated above.

4.2. A new randomized algorithm. Our goal in this section is to design an algorithm for multiset equality test for the modified model that is faster than ABK_1 for $n = \omega((\log\log a)^2)$ and faster than ABK_2 for $n = \omega\left((\log a)^{\log\log a}\right)$. We can then use it to speed up the previously best checking algorithm for sorting [3].

- Let $q = \lfloor \log a \rfloor + 1$.
- Let $x_1, \ldots, x_q$ be q distinct indeterminates.
- For each $u \in A \cup B$, let f_u denote the monomial $(x_1)^{u_1}(x_2)^{u_2}\cdots(x_q)^{u_q}$, where $u_1 u_2 \cdots u_q$ is the standard q-bit binary representation of u.
- Let $Q(x_1, \ldots, x_q)$ denote the polynomial $\sum_{i=1}^n f_{a_i} - \sum_{i=1}^n f_{b_i}$.

Note that $Q(x_1, \ldots, x_q) \equiv 0$ if and only if $A \equiv B$. To test whether $A \equiv B$, we detail how to implement the steps of Algorithm 1 to test Q as follows. The algorithm is analyzed only with respect to the modified model.

Remark. In the implementation, the parameter t of Theorem 2.3 needs to be a constant so that the algorithm can be performed inside the random access memory together with straightforward management of the tapes. At the end of this section, we set $t = 4$, but for the benefit of future research, we analyze the running time and the random bit count in terms of a general t.

Step 1. Compute q by finding the index of the most significant bit in the binary representation of a. Since a takes up $O(\frac{\log a}{\xi})$ words, this computation takes $O(q)$ time by shifting the most significant nonzero word to the left at most ξ times. Afterwards, set $d_1 = d_2 = \cdots = d_q = k_1 = k_2 = \cdots = k_q = k = 1$, $K = d = q$, $c = n$, and $Z = 2n$ in $O(q)$ time. This step takes $O(q)$ time.

Step 2. Compute the q smallest primes $p_{1,1}, p_{2,1}, \ldots, p_{q,1} \leq q\ln^2 q$. We compute these primes by inspecting $i = 2, 3, \ldots$ one at a time up to $q\ln^2 q$ until exactly q primes are found. Since i can fit into $O(1)$ words, it takes $O(\sqrt{q}\log q)$ time to check the primality of each i using the square root test for primes in a straightforward manner. Thus, this step takes $O(q^{3/2}\log^3 q)$ time.

Step 3. This step is straightforward and uses q random bits and $O(\frac{q}{\xi})$ time.

Step 4. Set $\ell = \lceil t \rceil \psi' + \lceil q \rceil + 1$, where t is a given positive number and $\psi' = 2\lceil \log n \rceil + \lceil \frac{q\lceil \log q \rceil}{2} \rceil + q\lceil \log\lceil \log q \rceil \rceil + 1$. The number $\lceil \log n \rceil$ can be computed from the input in $O(n)$ time. The computations of $\lceil \frac{\log q}{2} \rceil$ and $\lceil \log\lceil \log q \rceil \rceil$ are similar to Step 1 and take $O(\log q)$ time. Thus, this step takes $O(n + \log q + \frac{\log t}{\xi})$ time.

Step 5. As at Step 5 in section 3.2, we use Newton's method to compute $r_{i,1}$ for each $p_{i,1}$. With only integer operations allowed, we use $2^\ell g_j$ as the jth estimate for $2^\ell \sqrt{p_{i,1}}$; i.e., $2^\ell g_{j+1} = (2^\ell g_j + 2^{2\ell} p_{i,1}/(2^\ell g_j))/2$. The last estimate computed in this manner is $2^\ell r_{i,1}$. Since 2^ℓ can be computed in $O((\frac{\ell}{\xi})^2)$ time using a doubling process, the first estimate $2^\ell p_{i,1}$ can be computed in the same amount of time. Since the other estimates all are $O(\frac{\ell}{\xi})$ words long, the $(j + 1)$th estimate can be obtained from the jth in $O((\frac{\ell}{\xi})^2)$ time. Since only $O(\log \ell)$ iterations for each $2^\ell \sqrt{p_{i,1}}$ are needed, this

step takes $O(q(\frac{\ell}{\xi})^2 \log \ell)$ time.

Step 6. We compute $\Delta = Q((-1)^{b_{1,1}} r_{1,1}, \ldots, (-1)^{b_{q,1}} r_{q,1})$ by means of Fact 1 as follows. Let $\lambda = \lceil \log t \rceil$. Since $|2^{q\ell}\Delta|$ is an integer at most $2^{\psi'+q\ell}$, we uniformly and independently select λ random positive integers $w \leq (\psi'+q\ell)^2$ using $2\lambda(\log t + \log \log n + 2 \log \log a + o(\log \log a))$ random bits and $O(\lambda \frac{\log \ell}{\xi})$ time. Note that if $2^{q\ell}\Delta \neq 0$, then with probability at least $1 - \frac{1}{t}$, some $2^{q\ell}\Delta \bmod w$ is nonzero. We next compute all $2^{ql}\Delta \bmod w$. For each element $u \in A \cup B$, let $e(u)$ be the number of 0's in the standard q-bit binary representation of u. Let $h(u) = f_u((-1)^{b_{1,1}} 2^\ell r_{1,1}, \ldots, (-1)^{b_{q,1}} 2^\ell r_{q,1})$. Then, $2^{q\ell}\Delta = \sum_{i=1}^{n} 2^{e(a_i)\ell} h(a_i) - \sum_{i=1}^{n} 2^{e(b_i)\ell} h(b_i)$, which we use to compute all $2^{q\ell}\Delta \bmod w$ as follows.

- Compute the numbers $e(u)$ for all $u \in A \cup B$ in $O(nq)$ time.
- For all w, compute all $2^\ell r_{i,1} \bmod w$ in $O(\lambda q \frac{\ell}{\xi} \frac{\log \ell}{\xi})$ time.
- For all w, use values obtained above to compute $h(u) \bmod w$ for all u in $O(\lambda n q(\frac{\log \ell}{\xi})^2)$ time.
- For all w, compute $2^\ell \bmod w$ in $O(\lambda \frac{\ell}{\xi} \frac{\log \ell}{\xi})$ time.
- For all w, use values obtained above to compute $2^{e(u)\ell} \bmod w$ for all u in $O(\lambda n(\frac{\log \ell}{\xi})^2 \log q)$ time.
- For all w, use values obtained above to compute $2^{q\ell}\Delta \bmod w$ in $O(\lambda n(\frac{\log \ell}{\xi})^2)$ time.

This step uses $2\lambda(\log t + \log \log n + 2 \log \log a + o(\log \log a))$ random bits and takes $O(\lambda q \frac{\ell}{\xi} \frac{\log \ell}{\xi} + \lambda n q(\frac{\log \ell}{\xi})^2)$ time.

Step 7. Output "$A \not\equiv B$" if and only if some $2^{ql}\Delta \bmod w$ is nonzero.

The next theorem summarizes the above discussion.

THEOREM 4.1. *For any given $t > 2$, whether $A \equiv B$ can be determined in time*

$$O\left(q \log \ell \left(\frac{\ell}{\xi} \right)^2 + \lambda n q \left(\frac{\log \ell}{\xi} \right)^2 \right),$$

where $q = \Theta(\log a); \ell = \Theta(t(\log n + \log a \log \log a)); \xi = \Theta(\log \log(n+a)); \lambda = \Theta(\log t)$. The error probability is at most $\frac{2}{t}$ using $\log a + 2\lceil \log t \rceil (\log t + \log \log n + 2 \log \log a + o(\log \log a))$ random bits.

Proof. The running time of Algorithm 1 is dominated by those of Steps 5 and 6. The error probability follows from Theorem 2.3 and Fact 1. □

We use the next corollary of Theorem 4.1 to compare Algorithm 1 with ABK_1 and ABK_2 in the modified model.

COROLLARY 4.2. *With $t = 4$, Algorithm 1 has an error probability at most $\frac{1}{2}$ using $\log a + 4 \log \log n + 8 \log \log a + o(\log \log a)$ random bits, while running in time*

$$O\left(n \log a + \log a \frac{(\log n + \log a \log \log a)^2}{\log \log(n+a)} \right).$$

By Corollary 4.2, Algorithm 1 is faster than ABK_1 for $n = \omega((\log \log a)^2)$ and faster than ABK_2 for $n = \omega\left((\log a)^{\log \log a}\right)$. Thus, it can replace ABK_1 and ABK_2 to speed up the previously best checking algorithm for sorting [3] as follows. We use bucket sort for $2^a \leq n$, Algorithm 1 for $(\log a)^{\log \log a} \leq n < 2^a$, and ABK_2 otherwise.

REFERENCES

[1] M. Ajtai, J. Komlós, and E. Szemerédi, *Deterministic simulation in Logspace*, in Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, pp. 132–140.

[2] C. Berge, *Graphs*, second revised ed., North-Holland, Amsterdam, 1985.

[3] M. Blum and S. Kannan, *Designing programs that check their work*, J. ACM, 42 (1995), pp. 269–291.

[4] S. Chari, P. Rohatgi, and A. Srinivasan, *Randomness-optimal unique element isolation with applications to perfect matching and related problems*, SIAM J. Comput., 24 (1995), pp. 1036–1050.

[5] A. Cohen and A. Wigderson, *Dispersers, deterministic amplification, and weak random sources (extended abstract)*, in Proceedings of the 30th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1989, pp. 14–19.

[6] R. A. DeMillo and R. J. Lipton, *A probabilistic remark on algebraic program testing*, Inform. Process. Lett., 7 (1978), pp. 193–195.

[7] Z. Galil, S. Micali, and H. Gabow, *An $O(EV \log V)$ algorithm for finding a maximal weighted matching in general graphs*, SIAM J. Comput., 15 (1986), pp. 120–130.

[8] R. Impagliazzo and D. Zuckerman, *How to recycle random bits*, in Proceedings of the 30th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1989, pp. 248–253.

[9] N. Jacobson, *Basic Algebra*, W. H. Freeman, San Francisco, CA, 1974.

[10] R. M. Karp, E. Upfal, and A. Wigderson, *Constructing a perfect matching is in random NC*, Combinatorica, 6 (1986), pp. 35–48.

[11] W. J. LeVeque, *Topics in Number Theory*, Vol. 1, Addison-Wesley, Reading, MA, 1956.

[12] L. Lovasz, *On determinants, matchings and random algorithms*, in Fundamentals of Computing Theory (Proceedings of the Conference on Algebraic, Arith. and Categorical Methods in Comput. Theory, Berlin, Wendisch-Rietz, 1979), Math. Res. 2, L. Budach, ed., Akademia-Verlag, Berlin, 1979, pp. 565–574.

[13] S. Micali and V. V. Vazirani, *An $O(\sqrt{|V|} \cdot |E|)$ algorithm for finding maximum matching in general graphs*, in Proceedings of the 21st Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1980, pp. 17–27.

[14] P. Morandi, *Field and Galois Theory*, Grad. Texts in Math. 167, Springer-Verlag, New York, 1996.

[15] R. Motwani and P. Raghavan, *Randomized Algorithms*, Cambridge University Press, Cambridge, UK, 1995.

[16] V. Pan, *Complexity of parallel matrix computations*, Theoret. Comput. Sci., 54 (1987), pp. 65–85.

[17] J. H. Rowland and J. R. Cowles, *Small sample algorithms for the identification of polynomials*, J. ACM, 33 (1986), pp. 822–829.

[18] J. T. Schwartz, *Fast probabilistic algorithms for verification of polynomial identities*, J. ACM, 27 (1980), pp. 701–717.

[19] W. Thrash, *A Note on the Least Common Multiples of Dense Sets of Integers*, Tech. Report 93-02-04, Department of Computer Science, University of Washington, Seattle, WA, 1993.

[20] W. T. Tutte, *The factors of graphs*, Canad. J. Math., 4 (1952), pp. 314–328.

[21] V. V. Vazirani, *Maximum Matchings without Blossoms*, Ph.D. thesis, University of California, Berkeley, CA, 1984.

[22] R. E. Zippel, *Probabilistic algorithms for sparse polynomials*, in Proceedings of EUROSAM '79, an International Symposium on Symbolic and Algebraic Manipulation, Lecture Notes in Comput. Sci. 72, E. W. Ng, ed., Springer-Verlag, New York, 1979, pp. 216–226.