Reducing Randomness via Irrational Numbers

Zhi-Zhong Chen*
Computer Science Division
University of California at Berkeley
Berkeley, CA 94720-1776

Ming-Yang Kao†
Department of Computer Science
Duke University
Durham, NC 27708

Abstract

We propose a general methodology for testing whether a polynomial with integer coefficients is identically zero. The methodology is to evaluate the polynomial at suitable approximations of easily computable irrational points. An innovative feature of the methodology is that the error probability of the testing algorithm can be decreased by increasing the precision of the approximations of the chosen irrational numbers, instead of increasing the number of random bits as usual. To explain our methodology, we discuss the problem of deciding whether a graph has a perfect matching. Our new randomized NC algorithm uses fewer random bits without doing more work than all the previous randomized NC algorithms for the problem. We also apply the methodology to the problem of checking the equivalence of two multisets of integers. Our new randomized algorithm for this problem runs faster and uses fewer random bits than one of the two algorithms of Blum and Kannan. It also runs faster than the other for a large range of inputs on a more realistic model of computation than theirs.

1 Introduction

Many algorithms involve testing whether certain polynomials with integer coefficients are identically zero [3, 13, 16]. Often times, these polynomials have exponential-sized standard representations but have very succinct nonstandard representations. The determinant of the Tutte matrix of a graph is an example [15]. Here, we are interested in testing polynomials given in a succinct form.

1.1 Previous work and new methodology

Given a polynomial $Q(x_1, \ldots, x_m)$ in a succinct form, a naive method to test it is to transform it into the standard simplified form and then check whether its coefficients are all zero. This method takes too much time, because $Q(x_1, \ldots, x_m)$ may have an exponential number of monomials. Let d_Q be the degree of $Q(x_1, \ldots, x_m)$. An advanced method is to evaluate $Q(i_1, \ldots, i_m)$ [13, 16], where $i_1, \ldots, i_m$ are uniformly and independently chosen at random from a set S of $2d_Q$ integers. This method needs $m \cdot \lceil \log(2d_Q) \rceil$ random bits and achieves an error probability at most $\frac{1}{2}$. (Throughout this paper, all logarithms are base 2 unless explicitly specified otherwise.) There are three general techniques to use additional random bits to lower the error probability to $\frac{1}{t}$ for any integer $t > 2$. The first performs $\lceil \log t \rceil$ independent evaluations of Q at $\lceil \log(2d_Q) \rceil$-bit integers, using $m \cdot \lceil \log(2d_Q) \rceil \cdot \lceil \log t \rceil$ random bits. The second enlarges the size of the set S from $2d_Q$ to td_Q and performs one evaluation of Q at $\lceil \log d_Q + \log t \rceil$-bit integers, using $m \cdot \lceil \log d_Q + \log t \rceil$ random bits. The third is usually called *probability amplification* and works for $t \leq 2^{m\lceil \log(2d_Q) \rceil}$. It performs t pairwise independent evaluations of Q at $\lceil \log(2d_Q) \rceil$-bit integers, using $2m \cdot \lceil \log(2d_Q) \rceil$ random bits. Each of the three techniques has its own advantages and disadvantages in terms of the running time or the number of random bits used.

This paper proposes a new methodology for testing $Q(x_1, \ldots, x_m)$. (See §2.) Throughout this paper, we assume that the degree of $Q(x_1, \ldots, x_m)$ is polynomially bounded from above by the length of the input

---

*On leave from Department of Mathematical Sciences, Tokyo Denki University, Hatoyama, Saitama 350-03, Japan. Email: zchen@cs.berkeley.edu.

†Research supported in part by NSF Grant CCR-9101385. Email: kao@cs.duke.edu.

STOC '97 El Paso, Texas USA

representation of $Q(x_1, \ldots, x_m)$. Our methodology is to compute $Q(\pi_1, \ldots, \pi_m)$, where $\pi_1, \ldots, \pi_m$ are suitable irrational numbers such that $Q(\pi_1, \ldots, \pi_m) = 0$ if and only if $Q(x_1, \ldots, x_m)$ is identically zero. Since rational arithmetic must be used in any computer implementation, we replace each π_i with an approximate rational number π_i'. The question is how many bits each π_i' should have in order to guarantee that $Q(\pi_1', \ldots, \pi_m') = 0$ if and only if $Q(x_1, \ldots, x_m)$ is identically zero. We give an explicit answer to this question, from which we obtain a new randomized algorithm for testing $Q(x_1, \ldots, x_m)$. The algorithm runs in polynomial time and uses $\sum_{i=1}^{m} \lceil \log(d_i+1) \rceil$ random bits, where d_i is the degree of x_i in $Q(x_1, \ldots, x_m)$. Moreover, the error probability can be made inverse polynomially small by increasing the bit-length of each π_i'. Thus, compared with the previous techniques, our methodology has two advantages:

- The number of random bits used is smaller when some of $d_1, \ldots, d_m$ are smaller than d_Q.
- The error probability can be decreased without using a single additional random bit.

1.2 Application to perfect matching test

Our methodology is best explained by the problem of testing whether an n-vertex m-edge graph G has a perfect matching. (See §3.) Various sequential and parallel algorithms have been designed for this fundamental problem. It remains open whether this problem has a deterministic NC algorithm. The algorithm of Lovasz [8] performs the least work among all the previous randomized NC algorithms for this problem. His algorithm tests whether the Tutte matrix M of G is a nonzero polynomial by first replacing each indeterminate entry of M with a random integer in the set $\{1, 2, \ldots, 2n\}$ and then computing the determinant of the resulting numerical matrix. His algorithm achieves an error probability at most $\frac{1}{2}$, using $m \cdot \lceil \log(2n) \rceil$ random bits. The algorithm of Chari et al [4] uses the fewest random bits among all the previous randomized NC algorithms. Their algorithm achieves an error probability at most $\frac{3}{4}$, using $\min\{5m + 4\lceil \log Z \rceil, 24\lceil \log Z \rceil\} + O(\log n)$ random bits, where Z is any given upper bound on the number of perfect matchings in G. In general, G may have $\Pi_{i=1}^{n} d_i$ perfect matchings, where d_i is the degree of the i-th vertex v_i in G. Thus, their algorithm needs $\min\{24 \cdot \sum_{i=1}^{n} \lceil \log d_i \rceil, 5m + 4 \cdot \sum_{i=1}^{n} \lceil \log d_i \rceil\} + O(\log n)$ random bits. Also, their algorithm needs to compute the inverse and the determinant of an n by n matrix whose entries are $2n^7$-bit integers, and hence uses a large number of processors, albeit polynomial in n.

Our methodology yields a new randomized NC algorithm for testing G. Like Lovasz's algorithm [8], our algorithm tests whether the Tutte matrix M of G is a nonzero polynomial. The number of random bits used is only $\sum_{i=1}^{n} \lceil \log \max\{1, n_i\} \rceil$, where n_i is the number of neighbors v_j of the i-th vertex v_i in G such that $j > i$. This bound significantly improves Chari et al's [4]. The time and processor complexities of our algorithm are dominated by those of computing the determinant of an n by n matrix whose entries are $3n\lceil \log \frac{m}{n} \cdot \log^2 n \rceil$-bit integers. The error probability achieved is at most $\frac{1}{2}$. Furthermore, the error probability can be made inverse polynomially small by increasing the bit-length of the matrix entries without using a single additional random bit. For example, if we increase the bit-length of the entries to $2n^7$, i.e., as large as Chari et al's, then the error probability is $O(\frac{\log \frac{m}{n} \cdot \log^2 n}{n^6})$. Our algorithm can also be modified to perform the same amount of work as Lovasz's [8] by computing a numerical determinant of M modulo a reasonably small random integer. It suffices for the modulo integer to be $O(\log n)$-bit long. Hence, we need only $O(\log n)$ additional random bits to generate a desired modulo integer. In summary, our algorithm uses fewer random bits without doing more work than all the previous randomized NC algorithms for the perfect matching problem [4, 8].

1.3 Application to multiset equality test

To further demonstrate the methodology, we discuss the problem of checking whether two given multisets of integers are equivalent. (See §4.) To design a checker for sorting [3], Blum and Kannan proposed two randomized algorithms for solving this problem on a special model of computation. The model reflects many sorting scenarios more closely than the usual RAM model. One of their algorithms needs to select a random prime among the primes in the range $[1, 3a\lceil \log(n+1) \rceil]$, where n and a are respectively the number and the largest value of elements in the two given multisets. Their other algorithm needs to select a random prime among the primes in $[1, 3n\lceil \log(a + 2n) \rceil]$. How to select such random primes was not mentioned in their paper [3]. To the best of our knowledge, there are two possible methods for this task. One method computes all primes in the desired ranges, and then randomly selects one. With this method, their second algorithm would take more time than sorting itself, and so does their first algorithm in the typical case $n < a$ for sorting. The other method randomly generates an integer in the desired ranges, and then checks whether the generated integer is really a prime. The probability that the generated integer is a prime is $O(\frac{1}{\log a + \log\log n})$ (respectively, $O(\frac{1}{\log n + \log\log a})$) by the Prime Number Theorem [7]. Consequently, the second process of generating a random prime must repeat $O(\log a + \log\log n)$ (respectively, $O(\log n + \log\log a)$) expected times. But then, the algorithms use a large number of random bits. Although the number of ran-

dom bits used by the algorithms can be reduced to $O(\log a + \log\log n)$ (respectively, $O(\log n + \log\log a)$) as shown by Nisan and Zuckerman [9], the constants hidden by the big-O notations are not small. Thus, neither method for generating a random prime works well.

To overcome the above difficulty in Blum and Kannan's algorithms, we observe that it is unnecessary to use a random prime there. Instead, it suffices to select a slightly larger random integer. Then, in their model of computation, the modified algorithms run in time $O(n\log a)$ and $O\left(n\max\{1,(\frac{\log n}{\log a})^2\}\right)$, achieve error probabilities at most $\frac{1}{2}$ and $\frac{3}{4}$, and use $2\log(a+1)+2\log\log(n+1)$ and $3\log n+2\log\log(a+2n)+O(1)$ random bits, respectively.

Our methodology yields a further improved new randomized algorithm for multiset equality test. In the Blum and Kannan model, our algorithm runs in $O(n\log a)$ time and achieves an error probability at most $\frac{3}{4}$ using only $\lceil\log a\rceil + 2\lceil\log\log n\rceil + 4\lceil\log\log a\rceil + 2\lceil\log\log\log a\rceil + O(1)$ random bits, improving one of the modified versions of Blum and Kannan's algorithms. The algorithm can be modified to use only $\lceil\log a\rceil + \lceil\log\log n\rceil + 2\lceil\log\log a\rceil + \lceil\log\log\log a\rceil + O(1)$ random bits with a slightly longer running time.

Our algorithm also runs efficiently in a more realistic model than Blum and Kannan's model. In our model, each of the $O(\log n + \log a)$ words in the random access memory can hold an integer in the range $[1,\max\{\lceil\log n\rceil,\lceil\log a\rceil\}]$, while in their model each word can hold an integer in $[1,a]$. In our model of computation, sorting the given two multisets by comparisons (respectively, by radix) takes time $O(n\log n\cdot\frac{\log a}{\log\log n+\log\log a})$ (respectively, $O(n\log a\cdot\frac{\log a}{\log\log n+\log\log a})$). Blum and Kannan's algorithms take time $O\left(n\log a\cdot(\frac{\log a+\log\log n}{\log\log n+\log\log a})^2\right)$ and $O\left(n\cdot\frac{\log^2 n+\log a\cdot(\log n+\log\log a)}{(\log\log n+\log\log a)^2}\right)$, respectively. In contrast, our algorithm takes only $O\left(n\log a+\frac{\log^2 n\cdot\log a+\log^3 a\cdot(\log\log a)^2}{\log\log n+\log\log a}\right)$ time. Thus, our algorithm improves one of their algorithms in terms of both the running time and the number of random bits used. It runs faster than the other for $a = O(2^{\sqrt{n}/\log n})$ and uses fewer random bits for $a = O(\frac{n^3}{\log^4 n\cdot\log\log^2 n})$. Note that the range $[1,2^{\sqrt{n}/\log n}]$ is sufficiently large for practical purposes.

2 A generic algorithm for testing polynomials

Let $Q(x_1,\ldots,x_m)$ be a polynomial with integer coefficients. We want to test whether $Q=0$.

- For $i=1,2,\ldots,m$, let d_i be the degree of x_i in Q and $k_i=\lceil\log(d_i+1)\rceil$. Let $s=\sum_{i=1}^m k_i$.
- Let d be the degree of Q. Let c be the largest absolute value of a coefficient in Q.

Lemma 2.1 Let $q_{1,1},\ldots,q_{1,k_1},\ldots,q_{m,1},\ldots,q_{m,k_m}$ be s square-free positive integers which are pairwise relatively prime. Let $a_{1,1},\ldots,a_{1,k_1},\ldots,a_{m,1},\ldots,a_{m,k_m}$ be s bits. For $i=1,2,\ldots,m$, let $q_i=\sum_{j=1}^{k_i}(-1)^{a_{i,j}}\sqrt{q_{i,j}}$. Then, $Q(x_1,\ldots,x_m)$ is a nonzero polynomial if and only if $Q(q_1,\ldots,q_m)\neq 0$.

Proof. Let A_0 denote the field of rational numbers. For $j=1,\ldots,m$, let $s_j=\sum_{i=1}^j k_i$. Let A_j be the field generated by $q_1,q_2,\ldots,q_j$ over A_0. Let B_j be the field generated by $q_{1,1},\ldots,q_{1,k_1},\ldots,q_{j,1},\ldots,q_{j,k_j}$ over A_0. Note that $A_j=B_j$ and the dimension of A_j over A_0 is 2^{s_j}. In general, the dimension of A_j over A_{j-1} is 2^{k_j}. Thus, q_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, $Q(q_1,\ldots,q_j,x_{j+1},\ldots,x_m)$ is nonzero. The theorem is a special case of this statement with $j=m$. ∎

Lemma 2.2 Assume that $Q(x_1,\ldots,x_m)$ is not identically zero. Let $p_{1,1},\ldots,p_{1,k_1},\ldots,p_{m,1},\ldots,p_{m,k_m}$ be the smallest s primes. Let l be an integer $>\lceil\log(c+cd)\rceil+d\lceil\log m\rceil+2d\lceil\log s\rceil+d+1$. Let $r_{i,j}$, $1\le i\le m$ and $1\le j\le k_i$, be the rational number obtained from $\sqrt{p_{i,j}}$ by cutting off the bits that are less significant than the l-th bit after the decimal point. Let $b_{1,1},\ldots,b_{1,k_1},\ldots,b_{m,1},\ldots,b_{m,k_m}$ be s random bits. Then, with probability at least $1-\frac{\lceil\log(c+cd)\rceil+d\lceil\log m\rceil+2d\lceil\log s\rceil}{l-d-1}$, $Q\left(\sum_{j=1}^{k_1}(-1)^{b_{1,j}}r_{1,j},\ldots,\sum_{j=1}^{k_m}(-1)^{b_{m,j}}r_{m,j}\right)\neq 0$. Note that this probability can be increased solely by increasing l without using a single additional random bit.

Proof. For each sequence $a_{1,1},a_{1,2},\ldots,a_{1,k_1},\ldots,a_{m,1},a_{m,2},\ldots,a_{m,k_m}$ of s bits, we call $Q\left(\sum_{j=1}^{k_1}(-1)^{a_{1,j}}\sqrt{p_{1,j}},\ldots,\sum_{j=1}^{k_m}(-1)^{a_{m,j}}\sqrt{p_{m,j}}\right)$ a *conjugate* of $Q\left(\sum_{j=1}^{k_1}\sqrt{p_{1,j}},\ldots,\sum_{j=1}^{k_m}\sqrt{p_{m,j}}\right)$. Let Π be the product of the 2^s conjugates. By Lemma 2.1, $\Pi\neq 0$. Moreover, by algebra, Π is an integer. Thus, $|\Pi|\ge 1$.

Let $l'=l-\lceil\log(c+cd)\rceil-d\lceil\log m\rceil-2d\lceil\log s\rceil-d-1$. Let α be the number of conjugates of $Q\left(\sum_{j=1}^{k_1}\sqrt{p_{1,j}},\ldots,\sum_{j=1}^{k_m}\sqrt{p_{m,j}}\right)$ that are smaller than $2^{-l'}$. Let β be the number of the other conjugates. Then, $\alpha+\beta=2^s$. Each conjugate is the sum of at most $\sum_{i=0}^d m^i s^i$ terms, where the absolute value of each term is the product of a positive integer $\le c$ and at most d numbers among $\sqrt{p_{1,1}},\ldots,\sqrt{p_{1,k_1}},\ldots,\sqrt{p_{m,1}},\ldots,\sqrt{p_{m,k_m}}$. Thus, the absolute value of each conjugate does not exceed $c(d+1)m^d s^{2d}$, since each $\sqrt{p_{i,j}}\le\sqrt{s}\cdot\ln s$ by the Prime Number Theorem [7]. Thus, $\alpha\cdot(-l')+\beta\cdot(\log(c+cd)+d\log m+2d\log s)\ge 0$.

Therefore, $\frac{\beta}{2^s} \geq \frac{l'}{l'+\log(c+cd)+d\log m+2d\log s}$. This implies that with probability at least $\frac{l'}{l'+\log(c+cd)+d\log m+2d\log s}$, $|Q\left(\sum_{j=1}^{k_1}(-1)^{b_{1,j}}\sqrt{p_{1,j}},\ldots,\sum_{j=1}^{k_m}(-1)^{b_{m,j}}\sqrt{p_{m,j}}\right)| \geq 2^{-l'}$.

Now suppose that $|Q\left(\sum_{j=1}^{k_1}(-1)^{b_{1,j}}\sqrt{p_{1,j}},\ldots,\sum_{j=1}^{k_m}(-1)^{b_{m,j}}\sqrt{p_{m,j}}\right)| \geq 2^{-l'}$. We next show that the irrational numbers $\sqrt{p_{i,j}}$ can be truncated to sufficiently short rational numbers without reducing the value of Q substantially. To this end, first observe that $Q\left(\sum_{j=1}^{k_1}(-1)^{b_{1,j}}\sqrt{p_{1,j}},\ldots,\sum_{j=1}^{k_m}(-1)^{b_{m,j}}\sqrt{p_{m,j}}\right)$ is the sum of at most $\sum_{i=0}^{d} m^i s^i$ terms, where the absolute value of each term is the product of a positive integer $\leq c$ and at most d numbers among $\sqrt{p_{1,1}}$, ..., $\sqrt{p_{1,k_1}}$, ..., $\sqrt{p_{m,1}}$, ..., $\sqrt{p_{m,k_m}}$. If we cut off the bits of each $\sqrt{p_{i,j}}$ that are less significant than the l-th bit after the decimal point, then the absolute value of each term decreases by at most $2^{-l}c\left(\sqrt{s(\log s)^2}\right)^{d-1}(2^d-1)$. Thus, $|Q\left(\sum_{j=1}^{k_1}(-1)^{b_{1,j}}r_{1,j},\ldots,\sum_{j=1}^{k_m}(-1)^{b_{m,j}}r_{m,j}\right)| \geq |Q\left(\sum_{j=1}^{k_1}(-1)^{b_{1,j}}\sqrt{p_{1,j}},\ldots,\sum_{j=1}^{k_m}(-1)^{b_{m,j}}\sqrt{p_{m,j}}\right)| - c(d+1)m^d s^{2d}2^{-l+d}$. Since $l \geq \log(c+cd)+d\log m+2d\log s+d+1+l'$, we have $c(d+1)m^d s^{2d}2^{-l+d} < 2^{-l'-1}$. Hence, $|Q\left(\sum_{j=1}^{k_1}(-1)^{b_{1,j}}r_{1,j},\ldots,\sum_{j=1}^{k_m}(-1)^{b_{m,j}}r_{m,j}\right)|$ is no less than $2^{-l'-1}$ because $|Q\left(\sum_{j=1}^{k_1}(-1)^{b_{1,j}}\sqrt{p_{1,j}},\ldots,\sum_{j=1}^{k_m}(-1)^{b_{m,j}}\sqrt{p_{m,j}}\right)| \geq 2^{-l'}$.

By the above discussions, $|Q\left(\sum_{j=1}^{k_1}(-1)^{b_{1,j}}r_{1,j},\ldots,\sum_{j=1}^{k_m}(-1)^{b_{m,j}}r_{m,j}\right)| \geq 2^{-l'-1}$ with probability at least $\frac{l-(\log(c+cd)+d\log m+2d\log s+d+1)}{l-d-1}$. ∎

As shown in §3 and §4, the special case where each d_i is no more than 1 is very important. For this special case, a simple modification of the proof of Lemma 2.2 establishes the following corollary.

Corollary 2.3 Assume that $Q(x_1,\ldots,x_m)$ is not identically zero. Further suppose that the degree of each x_i in Q is at most 1 and that Q has at most Z monomials. Let p_1, ..., p_m be the smallest m primes. Let l be an integer $> \lceil\log Z\rceil+\lceil\log c\rceil+\lceil\frac{d\log m}{2}\rceil+d\lceil\log\log m\rceil+d+1$. Let r_i, $1 \leq i \leq m$, be the rational number obtained from $\sqrt{p_i}$ by cutting off the bits that are less significant than the l-th bit after the decimal point. Let b_1, ..., b_m be m random bits. Then, with probability at least $1-\frac{\lceil\log Z\rceil+\lceil\log c\rceil+\lceil\frac{d\log m}{2}\rceil+d\lceil\log\log m\rceil}{l-d-1}$, $Q\left((-1)^{b_1}r_1,\ldots,(-1)^{b_m}r_m\right) \neq 0$.

Lemma 2.2 suggests the following randomized algorithm for testing $Q(x_1,\ldots,x_m)$.

The generic algorithm

1. Compute the smallest s primes $p_{1,1}$, ..., p_{1,k_1}, ..., $p_{m,1}$, ..., p_{m,k_m}.
2. Choose a positive integer t. Set $l = t\cdot(\lceil\log(c+cd)\rceil+d\lceil\log m\rceil+2d\lceil\log s\rceil)+d+1$. Then, compute s rational numbers $r_{1,1}$, ..., r_{1,k_1}, ..., $r_{m,1}$, ..., r_{m,k_m}. Each $r_{i,j}$ is obtained from $\sqrt{p_{i,j}}$ by cutting off the bits that are less significant than the l-th bit after the decimal point.
3. For each $r_{i,j}$, set $r'_{i,j} = r_{i,j}$ or $-r_{i,j}$ with probability $\frac{1}{2}$, respectively.
4. Output "$Q(x_1,\ldots,x_m)$ is identically zero" if and only if $Q(\sum_{j=1}^{k_1} r'_{1,j},\ldots,\sum_{j=1}^{k_m} r'_{m,j}) = 0$.

Theorem 2.4 If $Q(x_1,\ldots,x_m)$ is identically zero, then the generic algorithm always outputs the correct answer; otherwise, it outputs the correct answer with probability at least $1-\frac{1}{t}$. Moreover, it uses exactly s random bits and its error probability can be decreased by increasing t without using a single additional random bit.

We postpone the efficiency issues of the generic algorithm to §5.

3 Testing perfect matchings in graphs

A *perfect matching* in a graph 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 some edge in L. The *perfect matching problem* is to decide whether a given graph has a perfect matching. Throughout this section, let $G=(V,E)$ be an n-vertex m-edge graph. Let $V=\{1,2,\ldots,n\}$. Without loss of generality, we assume that n is even and $m \geq \frac{n}{2}$.

The *Tutte matrix* of G is an n by n skew-symmetric matrix M of m distinct indeterminates $x_{i,j}$ whose (i,j)-th entry is $x_{i,j}$ if $\{i,j\} \in E$ and $i<j$; is $-x_{j,i}$ if $\{i,j\} \in E$ and $i>j$; and is 0 otherwise.

Theorem 3.1 (Tutte [15]) G has a perfect matching if and only if $\det M \neq 0$.

By this theorem, testing whether G has a perfect matching is equivalent to testing whether $\det M$ is a nonzero polynomial. It is not obvious how to test $\det M$, because $\det M$ may have an exponential number of monomials. The next theorem allows this test to be performed efficiently with randomization.

Theorem 3.2 (1978, Schwartz [13], Zippel [16]) Let $Q(y_1,\ldots,y_m)$ be a polynomial with integer coefficients and degree d_Q. Let S be a set of $2d_Q$ integers. If Q is not identically zero, then $Q(r_1,\ldots,r_m) \neq 0$ with probability at least $\frac{1}{2}$, where r_1, ..., r_m are uniformly and independently chosen at random from S.

The set S in Theorem 3.2 is usually chosen to be $\{1, 2, \ldots, 2d_Q\}$. Theorems 3.1 and 3.2 together yield a randomized NC algorithm for testing whether G has a perfect matching. Since the degree of $\det M$ is n whenever G has a perfect matching, the number of random bits used by the algorithm is $m \cdot \lceil \log(2n) \rceil$. The time and processor complexities of the algorithm are dominated by those of computing the determinant of an n by n matrix with $O(\log n)$-bit integer entries.

To apply our new methodology to testing $\det M$, we begin by reviewing an expansion of $\det M$ in terms of perfect matchings.

- Let $L = \{\{i_1, j_1\}, \ldots, \{i_{\frac{n}{2}}, j_{\frac{n}{2}}\}\}$ be an arbitrary perfect matching in G with $i_1 < j_1, i_2 < j_2, \ldots, i_{\frac{n}{2}} < j_{\frac{n}{2}}$ and $i_1 < i_2 < \cdots < i_{\frac{n}{2}}$. Let $\pi(L) = x_{i_1,j_1} x_{i_2,j_2} \cdots x_{i_{\frac{n}{2}},j_{\frac{n}{2}}}$. Let $\sigma(L) = 1$ or -1 if the following permutation is even or odd, respectively: $\begin{pmatrix} 1 & 2 & \cdots & n-1 & n \\ i_1 & j_1 & \cdots & i_{\frac{n}{2}} & j_{\frac{n}{2}} \end{pmatrix}$
- $\mathrm{Pf}(M) = \sum_L \pi(L) \cdot \sigma(L)$, where L ranges over all perfect matchings in G.

Theorem 3.3 (1961, Fisher and Kasteleyn [1]) $\det M = (\mathrm{Pf}(M))^2$.

Let G' be the acyclic digraph obtained from G by replacing each edge $\{i, j\}$ with the arc $(\min\{i, j\}, \max\{i, j\})$. For each vertex i in G', let n_i be the number of outgoing arcs of vertex i. Let $\delta_i = 0$ if $n_i = 0$; otherwise, $\delta_i = \lceil \log n_i \rceil$. Let $k = \sum_{i=1}^{n} \delta_i$. Note that $k < n + n \log \frac{m}{n}$. Let $y_1, y_2, \ldots, y_k$ be k distinct new indeterminates.

We label the n_1 outgoing arcs of vertex 1 in G' as follows. If $n_1 = 0$, vertex 1 has no outgoing arc in G'. If $n_1 = 1$, then we label its unique outgoing arc with 1. If $n_1 \geq 2$, then we label its n_1 outgoing arcs each with a distinct monomial in the set $\{(y_1)^{a_1} (y_2)^{a_2} \cdots (y_{\delta_1})^{a_{\delta_1}} \mid \text{each } a_i \text{ is 0 or 1}\}$, which is always possible since $2^{\delta_1} \geq n_1$. We label the n_2 outgoing arcs of vertex 2 in the same manner except that we use the indeterminates $y_{\delta_1+1}, y_{\delta_1+2}, \ldots, y_{\delta_1+\delta_2}$ instead of $y_1, y_2, \ldots, y_{\delta_1}$. We similarly label the outgoing arcs of the other vertices each time using indeterminates that have not been used for previous vertices.

Lemma 3.4 Let $f_{i,j}$ denote the label of each arc (i, j) in G'. Let $Q(y_1, \ldots, y_k)$ be the polynomial obtained from $\mathrm{Pf}(M)$ by replacing each indeterminate $x_{i,j}$ by $f_{i,j}$. Then, G has a perfect matching if and only if $Q(y_1, \ldots, y_k)$ is not identically zero.

Proof. By Theorems 3.1 and 3.3, G has a perfect matching if and only if $\mathrm{Pf}(M)$ is not identically zero. Thus, if G has no perfect matching, then $\mathrm{Pf}(M)$ is a zero polynomial and so is $Q(y_1, \ldots, y_k)$ by the definition of Q.

We next show that if G has a perfect matching, then $Q(y_1, \ldots, y_k)$ is not identically zero. First note that there is a one-to-one onto correspondence between the perfect matchings in G and the monomials in Q. A perfect matching $L = \{\{i_1, j_1\}, \{i_2, j_2\}, \ldots, \{i_{\frac{n}{2}}, j_{\frac{n}{2}}\}\}$ with $i_1 < j_1$, ..., $i_{\frac{n}{2}} < j_{\frac{n}{2}}$ corresponds to the monomial $\sigma(L) \cdot f_{i_1,j_1} f_{i_2,j_2} \cdots f_{i_{\frac{n}{2}},j_{\frac{n}{2}}}$, which we denote by Q_L. Moreover, $Q = \sum_L Q_L$, where L ranges over all the perfect matchings in G.

Note that for every perfect matching L, each indeterminate y_i has a degree at most 1 in Q_L. If Q_L has a degree at least 1, then $|Q_L((-1)^{b_1}\sqrt{p_1}, \ldots, (-1)^{b_k}\sqrt{p_k})|$ is the product of one or more distinct indeterminates among $y_1, \ldots, y_k$. Otherwise, $|Q_L| = 1$.

Let L_1 and L_2 be two distinct perfect matchings in G. Let H be the subgraph of G induced by the edges in $(L_1 \cup L_2) - (L_1 \cap L_2)$. H is a collection of vertex-disjoint cycles. Since $L_1 \neq L_2$, H contains at least one cycle C. Let C' be the digraph obtained from C by replacing each edge $\{i, j\}$ by the arc $(\min\{i, j\}, \max\{i, j\})$. C' is a subgraph of G' and is not a directed cycle. In C', there is a vertex i such that the two arcs incident to i are both outgoing arcs of i. Let (i, j_1) and (i, j_2) be the outgoing arcs of i. Then, there is an indeterminate y_t, $\delta_{i-1} + 1 \leq t \leq \delta_i$, whose degree is 1 in one of the two monomials f_{i,j_1} and f_{i,j_2} but is 0 in the other. Hence, the degree of y_t is 1 in one of the two monomials Q_{L_1} and Q_{L_2} but is 0 in the other. Thus, Q_{L_1} and Q_{L_2} are two distinct monomials. Therefore, the number of distinct monomials in Q is the same as the number of perfect matchings in G. This establishes the lemma. ∎

Lemma 3.5 Assume that $Q(y_1, \ldots, y_k)$ is not identically zero. Let $p_1, \ldots, p_k$ be the smallest k primes. Let l be an integer $> n\lceil \log \frac{2m}{n} \rceil + \lceil \frac{k \log k}{2} \rceil + k\lceil \log \log k \rceil + k + 1$. Let r_i, $1 \leq i \leq k$, be the rational number obtained from $\sqrt{p_i}$ by cutting off the bits that are less significant than the l-th bit after the decimal point. Let $b_1, \ldots, b_k$ be k random bits. Then, with probability at least $1 - \frac{n\lceil \log \frac{2m}{n} \rceil + \lceil \frac{k \log k}{2} \rceil + k\lceil \log \log k \rceil}{l-k-1}$, $Q\left((-1)^{b_1} r_1, \ldots, (-1)^{b_k} r_k\right) \neq 0$.

Proof. Let Z be the number of perfect matchings in G. As observed in [4], $Z < (\frac{2m}{n})^n$. Moreover, by the proof of Lemma 3.4, Q has exactly Z monomials, the degree of each indeterminate y_i in Q is at most 1, and the largest absolute value of a coefficient in Q is no more than 1. Also, the degree of Q does not exceed k. All these facts together with Corollary 2.3 imply the lemma. ∎

Lemma 3.5 suggests the following randomized algorithm for testing whether G has a perfect matching.

Algorithm 1

1. Compute the smallest k primes $p_1, \ldots, p_k$.

2. Choose a positive integer t. Set $l = t \cdot \left(n\lceil\log\frac{2m}{n}\rceil + \lceil\frac{k\log k}{2}\rceil + k\lceil\log\log k\rceil\right) + k + 1$. Then, compute the square roots of $p_1, \ldots, p_k$ up to the l-th bit after the decimal point to obtain k rational numbers $r_1, \ldots, r_k$.

3. Generate k random bits $b_1, \ldots, b_k$ and compute $\left(Q((-1)^{b_1}r_1, \ldots, (-1)^{b_k}r_k)\right)^2$ by computing $\det M$ at the points $(-1)^{b_1}r_1, \ldots, (-1)^{b_k}r_k$.

4. Output "G has a perfect matching" if and only if $\left(Q((-1)^{b_1}r_1, \ldots, (-1)^{b_k}r_k)\right)^2 \neq 0$.

When G has no perfect matching, Algorithm 1 always outputs the correct answer. When G has a perfect matching, Algorithm 1 outputs the correct answer with probability at least $1 - \frac{1}{t}$.

For Algorithm 1 to work, we need to answer some questions. How do we generate $p_1, \ldots, p_k$? By the Prime Number Theorem [7], for some positive constant $\epsilon \leq 1$, there are at least k primes $p_1, \ldots, p_k$ among 2, 3, $\ldots$, $k\cdot(\ln k)^{1+\epsilon}$. To find $p_1, \ldots, p_k$, we use the fact that $p \in \{2, 3, \ldots, k\cdot(\ln k)^{1+\epsilon}\}$ is a prime if and only if p is indivisible by any integer q with $2 \leq q \leq \lfloor\sqrt{p}\rfloor$. The desired primes can be found in $O(\log n)$ parallel arithmetic steps with $O\left(k^{1.5}(\ln k)^{1.5+1.5\epsilon}\right)$ processors. They can also be found in $O(\log n)$ parallel Boolean steps with $O\left(k^{1.5}(\ln k)^{1.5+1.5\epsilon}\log n\cdot\log\log\log n\right)$ processors, since division of $O(\log n)$-bit integers takes $O(\log\log n\cdot\log\log\log n)$ parallel Boolean steps with $O(\log n\cdot\log\log\log n)$ processors [12].

To compute r_i from each p_i, we use Newton's method. For completeness, we briefly sketch the method as follows. To compute r_i, we start with $g_0 = p_i$ as the initial estimate. Suppose that we have computed the j-th estimate g_j with $j \geq 0$. By Newton's method, $g_{j+1} = \frac{1}{2}(g_j + \frac{p_i}{g_j})$. To compute g_{j+1} from g_j using this equation, we maintain only those bits of g_{j+1} that are more significant than the $(l+1)$-th bit after the decimal point. Thus, we have $g_{j+1} \leq \frac{1}{2}(g_j + \frac{p_i}{g_j})$ in general. Once we obtain g_{j+1}, we check whether $g_{j+1}^2 > p_i$. If not, we stop; otherwise, we proceed to compute g_{j+2}. One can verify that $g_{j+1} - \sqrt{p_i} \leq \frac{(g_j - \sqrt{p_i})^2}{2g_j}$. Since we maintain $g_j > \sqrt{p_i}$ except the last j, the convergence order of the sequence $g_0, g_1, \ldots$ is 2. Therefore, we can take the $\lceil\log(\lceil\log p_i\rceil + l)\rceil$-th estimate as r_i. Obviously, $r_1, \ldots, r_k$ can be computed in $O\left(\log(l + \log k)\right)$ parallel arithmetic steps with k processors. They can also be computed in $O\left(\log^2(l+\log k)\cdot\log\log(l+\log k)\right)$ parallel Boolean steps with $O\left(k(l+\log k)\cdot\log\log(l+\log k)\right)$ processors, since division of $O(l + \log k)$-bit numbers takes $O\left(\log(l+\log k)\cdot\log\log(l+\log k)\right)$ parallel Boolean steps with $O\left((l+\log k)\cdot\log\log(l+\log k)\right)$ processors [12].

We can evaluate $\left(Q((-1)^{b_1}r_1, \ldots, (-1)^{b_k}r_k)\right)^2$ in terms of operations on integers as follows. By its definition, $\left(Q((-1)^{b_1}r_1, \ldots, (-1)^{b_k}r_k)\right)^2$ is the determinant of an n by n skew-symmetric matrix whose nonzero entries above the main diagonal in the i-th row each are either 1 or the product of at most δ_i numbers among $r_1, \ldots, r_k$. For each row i and each column j of the matrix, we multiply $2^{(\delta_i+\delta_j)l}$ to the (i,j)-th entry. This takes $O(1)$ parallel arithmetic steps with $O(n^2)$ processors or $O(\log l)$ parallel Boolean steps with $O(n^2 l\log n)$ processors. The determinant of the matrix then increases by a factor of 2^{2kl}. Moreover, the (i,j)-th entry is then an integer with at most $(\delta_i+\delta_j)\cdot(\lceil\frac{\log k}{2}\rceil + \lceil\log\log k\rceil + l)$ bits. Each entry of the matrix is hence at most $2\lceil\log n\rceil(l + \lceil\log n\rceil)$-bit long, since $k \leq n\log\frac{2m}{n}$. Therefore, instead of computing $\left(Q((-1)^{b_1}r_1, \ldots, (-1)^{b_k}r_k)\right)^2$ in step 3, we can compute the determinant of a matrix with $2\lceil\log n\rceil(l + \lceil\log n\rceil)$-bit integer entries.

Note that l can be as large as any integer polynomial in n without violating the goal that Algorithm 1 runs in poly-logarithmic time using a polynomial number of processors. Thus, the error probability of Algorithm 1 can be made inverse polynomially small by increasing l.

Let $T_{\det}^A(h)$ and $P_{\det}^A(h)$ be the numbers of parallel arithmetic steps and processors to compute the determinant of an h by h integer matrix whose entries each have a bit-length polynomial in h. Currently, $T_{\det}^A(h) = O(\log^2 h)$ and $P_{\det}^A(h) = O(h^{2.376})$ [11]. Similarly, let $T_{\det}^B(h,b)$ and $P_{\det}^B(h,b)$ be the numbers of parallel Boolean steps and processors to compute the determinant of an h by h matrix whose entries are b-bit integers. Currently, $T_{\det}^B(h,b) = O(\log h\cdot\log d)$ and $P_{\det}^B(h,b) = O(h^{2.86}d\log d\cdot\log\log d)$ [11], where $d = hb + h\log h$. Summarizing the above discussions in this section, we obtain the following theorem.

Theorem 3.6 Let l be an arbitrary integer larger than $n\lceil\log\frac{2m}{n}\rceil + \lceil\frac{k\log k}{2}\rceil + k\lceil\log\log k\rceil + k + 1$. Then, Algorithm 1 runs in $T_{\det}^A(n) + O(\log l)$ parallel arithmetic steps with $P_{\det}^A(n) + O(n^2)$ processors or in $T_{\det}^B(n, 2l\lceil\log n\rceil + 2\lceil\log n\rceil^2) + O(\log^2 l\cdot\log\log l)$ parallel Boolean steps with $P_{\det}^B(n, 2l\lceil\log n\rceil + 2\lceil\log n\rceil^2) + O\left(nl\log n\cdot(n + \log\log l)\right)$ processors. Moreover, it uses exactly k random bits to guarantee an error probability at most $\frac{n\lceil\log\frac{2m}{n}\rceil + \lceil\frac{k\log k}{2}\rceil + k\lceil\log\log k\rceil}{l-k-1}$.

By Theorem 3.6, to guarantee an error probability at most $\frac{1}{2}$, it suffices to set $l = 2\cdot(n\lceil\log\frac{2m}{n}\rceil + \lceil\frac{k\log k}{2}\rceil + k\lceil\log\log k\rceil) + k + 1$ in Algorithm 1. In contrast, to guarantee an error probability at most $\frac{3}{4}$, the algorithm in [4] computes the determinant of a matrix with $O(n^7)$-bit integer entries and uses $\min\{24\cdot\sum_{i=1}^n\lceil\log d_i\rceil,\ 5m + 4\cdot\sum_{i=1}^n\lceil\log d_i\rceil\} + O(\log n)$ random bits, where d_i is the degree of vertex i in G. Thus, Algorithm 1 uses fewer random bits and performs less work than the algorithm in [4].

Recall that Theorems 3.1 and 3.2 together yield a randomized NC algorithm for the perfect matching problem. To guarantee an error probability at most $\frac{1}{t}$ for $t \geq 2$, this algorithm uses $m \log(tn)$ random bits. In contrast, Algorithm 1 uses fewer random bits to guarantee the same error probability. However, the algorithm implied by Theorems 3.1 and 3.2 runs in $T^B_{\det}(n, \lceil \log(tn) \rceil) + O(\log n)$ parallel Boolean steps with $P^B_{\det}(n, \lceil \log(tn) \rceil) + O(n^2)$ processors, and hence performs less work than Algorithm 1. Can we modify Algorithm 1 so that it needs to compute the determinant of only a matrix with $O(\log n)$-bit integer entries? The rest of this section is devoted to answering this question.

Theorem 3.7 (Thrash [14]) Let $h \geq 3$. Let H be a subset of $\{1, 2, \ldots, h^2\}$ such that $|H| \geq \frac{h^2}{2}$. Then, the least common multiple of the elements in H exceeds 2^h.

By Theorem 3.7 and the Chinese Remainder Theorem, for any positive integer $h' \leq 2^h$, an integer randomly chosen from $\{1, 2, \ldots, h^2\}$ does not divide h' with probability at least $\frac{1}{2}$. Consequently, if we choose γ integers uniformly and independently at random from $\{1, 2, \ldots, h^2\}$, then at least one of them does not divide h' with probability at least $1 - 2^{-\gamma}$.

Theorem 3.8 Let l be the same as in Theorem 3.6. Then, for any positive integer γ, Algorithm 1 can be modified to run in $\gamma \cdot T^B_{\det}(n, O(\log l)) + O(\log^2 l \cdot \log\log l)$ parallel Boolean steps with $P^B_{\det}(n, O(\log l)) + \gamma + O(nl \log n \cdot (n + \log\log l))$ processors, using $k + O(\gamma \log l)$ random bits to guarantee an error probability at most $\frac{n\lceil \log \frac{2m}{n} \rceil + \lceil \frac{k \log k}{2} \rceil + k\lceil \log\log k \rceil}{l-k-1} + 2^{-\gamma}$.

Proof. Recall that Algorithm 1 needs to compute the determinant of a matrix M' whose entries are $2(l + \lceil \log n \rceil)\lceil \log n \rceil$-bit integers. Clearly, $|\det M'| \leq n! \cdot 2^{2n(l+\lceil \log n \rceil)\lceil \log n \rceil}$. Let γ be a positive integer. By Theorem 3.7, if we select γ integers uniformly and independently at random among the smallest $O(n^2 l^2 \log^2 n)$ positive integers, then with probability at least $1 - 2^{-\gamma}$, at least one of the chosen integers does not divide $\det M'$ whenever $\det M' \neq 0$. The desired γ random integers can be chosen using only $O(\gamma \log l + \gamma \log n)$ additional random bits in $O(\log l + \log n)$ parallel Boolean steps with γ processors. Instead of computing $\det M'$ in Algorithm 1, we can compute $\det M'$ modulo each of the chosen γ integers and report "G has a perfect matching" if and only if at least one of the γ remainders is not zero. The overall error probability is less than $\frac{n\lceil \log \frac{2m}{n} \rceil + \lceil \frac{k \log k}{2} \rceil + k\lceil \log\log k \rceil}{l-k} + 2^{-\gamma}$. ∎

By Theorem 3.8, to guarantee an error probability at most $\frac{1}{t}$ for an integer $t \geq 2$, it suffices to set $\gamma = \lceil \log(2t) \rceil$ and $l = 2t \cdot (n\lceil \log \frac{2m}{n} \rceil + \lceil \frac{k \log k}{2} \rceil + k\lceil \log\log k \rceil) + k$. Then, the modified version of Algorithm 1 runs in $\lceil \log(2t) \rceil \cdot T^B_{\det}(n, O(\log t + \log n)) + O\left(\log^2(t+n) \cdot \log\log(t+n)\right)$ parallel Boolean steps with $P^B_{\det}(n, O(\log t + \log n)) + O\left(tn^2 \log^3 n \cdot (n + \log\log t)\right)$ processors, using $k + O(\log t \cdot \log n + \log^2 t)$ random bits. Recall that $T^B_{\det}(n, O(\log n)) = O(\log^2 n)$ and $P^B_{\det}(n, O(\log n)) = O(n^{3.86} \log^2 n \cdot \log\log n)$ [11] at present. Thus, if we maintain $t = O(n^{0.86})$, then the modified version performs almost the same amount of work as the algorithm implied by Theorems 3.1 and 3.2 but uses much fewer random bits.

4 Multiset equality test

The *multiset equality test problem* is that of deciding whether two given multisets of integers are equivalent. Blum and Kannan gave two algorithms for this problem [3]. Their algorithms are designed for a special model of computation that reflects many sorting scenarios more closely than the usual RAM model. Here, we employ an even more realistic model of computation.

We fix two multisets $S = \{s_1, \ldots, s_n\}$ and $T = \{t_1, \ldots, t_n\}$ of positive integers. Let a be the largest possible value for any element of $S \cup T$.

4.1 Model of computation and previous results

In our model of computation, the computer has a fixed number of tapes, including one that contains S and another that contains T. S and T each have at most n elements, and each element is an integer in $\{1, 2, \ldots, a\}$. The random access memory has $O(\log n + \log a)$ words each of which can hold an integer in the range $[1, \max\{\lceil \log n \rceil, \lceil \log a \rceil\}]$. The allowed elementary operations are $+$, $-$, $\times$, $/$, $<$, $=$, and two bit operations *shift-to-left* and *shift-to-right*. Here, $/$ is "integer divide". Each of the operations takes one step on integers that are one word long. On integers that are m words long, $\times$ and $/$ each take m^2 steps, and the other operations each take m steps. In addition, each shift of a tape takes one step. Each copy of a word on tape to a word in the random access memory or vice versa takes one step.

In our model of computation, when $n \geq 2^a$, whether S and T are equivalent can be decided in optimal $O(n)$ time by using bucket sort. Hence, we hereafter assume $n < 2^a$. Note that the only difference between our model and Blum and Kannan's is that our model has shorter word length. In Blum and Kannan's model, each word can hold an integer in the range $[1, a]$.

In our model, sorting S and T by comparisons takes $O(n \log n \cdot \frac{\log a}{\log\log n + \log\log a})$ time, and sorting them by radix takes $O(n \log a \cdot \frac{\log a}{\log\log n + \log\log a})$ time. We briefly describe the two algorithms of Blum and Kannan for

multiset equality test [3]. One algorithm selects a random prime p uniformly from the primes in the range $[1, 3a\lceil\log(n+1)\rceil]$, and then checks whether $\sum_{i=1}^{n}(n+1)^{s_i} \equiv \sum_{i=1}^{n}(n+1)^{t_i} \bmod p$. Assuming that p is available in one step, this algorithm takes $O(n \log a)$ time in their model, but takes $O\left(n \log a \cdot (\frac{\log a}{\log\log n + \log\log a})^2\right)$ time in our model since p is $O(\log a)$-bit long. Their other algorithm tests whether the polynomial $Q(x) = \Pi_{i=1}^{n}(x-s_i) - \Pi_{i=1}^{n}(x-t_i)$ is identically zero. To do this, it uniformly selects a random integer $z \in \{1, 2, \ldots, 2n\}$ and a random prime q from the primes in the range $[1, 3n\lceil\log(a+2n)\rceil]$, and then checks whether $Q(z) \equiv 0 \bmod q$. Assuming that q is available in one step, this algorithm takes $O\left(n \max\{1, (\frac{\log n}{\log a})^2\}\right)$ time in their model, but takes $O\left(n \cdot \frac{(\log n + \log a)(\log n + \log\log a)}{(\log\log n + \log\log a)^2}\right)$ time in our model since computing $(z - s_1) \bmod q$, ..., $(z - s_n) \bmod q$, $(z - t_1) \bmod q$, ..., $(z - t_n) \bmod q$ takes this much time.

Generating large random primes is a crucial step of Blum and Kannan's algorithms. Here, we show that this step can be avoided by using Theorem 3.7. Recall that their algorithms check whether $\sum_{i=1}^{n}(n+1)^{s_i} = \sum_{i=1}^{n}(n+1)^{t_i}$ and whether $Q(x)$ is a nonzero polynomial, respectively. One can verify that $|\sum_{i=1}^{n}(n+1)^{s_i} - \sum_{i=1}^{n}(n+1)^{t_i}| \leq 2^{a\log(n+1)+\log n}$ and $Q(2n) \leq 2^{n\log(a+2n)}$. Therefore, we can replace the random primes p and q in their algorithms with two random integers $p' \in \{1, 2, \ldots, (a\log(n+1) + \log n)^2\}$ and $q' \in \{1, 2, \ldots, (n\log(a+2n))^2\}$, respectively. Generating p' and q' is trivial, and takes $2\log(a+1) + 2\log\log(n+1)$ and $2\log n + 2\log\log(a+2n)$ random bits, respectively.

4.2 A new randomized algorithm

We now present a new randomized algorithm for multiset equality test. Recall that $S = \{s_1, \ldots, s_n\}$, $T = \{t_1, \ldots, t_n\}$, and $S \cup T \subseteq \{1, 2, \ldots, a\}$. Let $k = \lfloor\log a\rfloor + 1$. Let x_1, ..., x_k be k distinct indeterminates. For each $u_i \in S \cup T$, let $f(u_i)$ denote the monomial $(x_1)^{a_1}(x_2)^{a_2}\cdots(x_k)^{a_k}$, where $a_1 a_2 \cdots a_k$ is the standard k-bit representation of u_i. Define a polynomial $P(x_1, \ldots, x_k) = \sum_{i=1}^{n} f(s_i) - \sum_{i=1}^{n} f(t_i)$. Note that $P(x_1, \ldots, x_k)$ is identically zero if and only if S and T are equivalent.

Lemma 4.1 Assume that P is not identically zero. Let p_1, ..., p_k be the smallest k primes. Let l be an integer $> \lceil\log n\rceil + \lceil\frac{k\log k}{2}\rceil + k\lceil\log\log k\rceil + k + 2$. Let r_i, $1 \leq i \leq k$, be the rational number obtained from $\sqrt{p_i}$ by cutting off the bits that are less significant than the l-th bit after the decimal point. Let b_1, ..., b_k be k random bits. Then, with probability at least $1 - \frac{1+\lceil\log n\rceil+\lceil\frac{k\log k}{2}\rceil+k\lceil\log\log k\rceil}{l-k-1}$, $P\left((-1)^{b_1}r_1, \ldots, (-1)^{b_s}r_s\right) \neq 0$.

Proof. By the definition of $P(x_1, \ldots, x_k)$, P has at most $2n$ monomials, the degree of each indeterminate x_i in P is at most 1, and the largest absolute value of a coefficient in P is at most 1. Also, the degree of P does not exceed k. All these facts together with Corollary 2.3 establishes the lemma. ∎

Lemma 4.1 suggests the following randomized algorithm for testing $P(x_1, \ldots, x_k)$.

Algorithm 2

1. Compute the smallest k primes p_1, ..., p_k and generate k random bits b_1, ..., b_k.
2. Set $l = 2 \cdot \left(1 + \lceil\log n\rceil + \lceil\frac{k\log k}{2}\rceil + k\lceil\log\log k\rceil\right) + k + 1$, and compute the square roots of p_1, ..., p_k up to the l-th bit after the decimal point to obtain k rational numbers r_1, ..., r_k.
3. Select a random integer q (to be specified later) and compute $2^{kl}P\left((-1)^{b_1}r_1, \ldots, (-1)^{b_k}r_k\right) \bmod q$.
4. Output "S and T are equivalent" if and only if the value obtained at step 3 is 0.

We next discuss how to perform steps 1, 2, and 3 efficiently in our model of computation.

Step 1. Recall that $k = \lfloor\log a\rfloor + 1$. We can compute a in $O(n \cdot \frac{\log a}{\log\log n + \log\log a})$ time. We can obtain k from a in $O(\log a)$ time. By the Prime Number Theorem, p_1, ..., p_k are in the set $C = \{2, 3, \ldots, k(\ln k)^2\}$. To obtain these k primes, we repeatedly inspect the elements of C in increasing order until we find exactly k primes. We record the primes found in the process on a single tape. Since each element in C is only $O(\log\log a)$-bit long, the whole process takes $O\left((k(\ln k)^2)^{3/2}\right) = O\left((\log a)^{3/2} \cdot (\log\log a)^3\right)$ time. Thus, step 1 takes $O\left(n \cdot \frac{\log a}{\log\log n + \log\log a} + (\log a)^{3/2} \cdot (\log\log a)^3\right)$ time.

Step 2. To compute each r_i, we again use Newton's method, but instead of using $g_{j+1} = \frac{1}{2}(g_j + \frac{p_i}{g_j})$ to compute g_{j+1}, we use the equation $2^l g_{j+1} = (2^l g_j + 2^{2l}p_i/(2^l g_j))/2$ to compute $2^l g_i$. Here, / is "integer divide". The final estimate computed in this manner is $2^l r_i$. We record $2^l r_1$, ..., $2^l r_k$ on a single tape. Step 2 takes $O\left(\log a \cdot \frac{(\log n + \log a \log\log a)^2}{\log\log n + \log\log a}\right)$ time.

Step 3. To see how small q can be, note that $2^{kl}P\left((-1)^{b_1}r_1, \ldots, (-1)^{b_k}r_k\right) \leq 2^{kl+1}n(\sqrt{k}\ln k)^k$. Thus, by Theorem 3.7, we may select q uniformly at random from $\{1, 2, \ldots, (1 + \log n + \frac{k\log k}{2} + k\log\log k + kl)^2\}$. Then, with probability at least $\frac{1}{2}$, $2^{kl}P\left((-1)^{b_1}r_1, \ldots, (-1)^{b_k}r_k\right) \bmod q$ is not 0 whenever $2^{kl}P\left((-1)^{b_1}r_1, \ldots, (-1)^{b_k}r_k\right) \neq 0$. Selecting q takes $O(\log\log n + \log\log a)$ time and uses at most $2\log\log n + 4\log\log a + 2\log\log\log a + O(1)$ random

bits. An alternative way to select q is to select it uniformly at random from the smallest $2(1 + \log n + \frac{k \log k}{2} + k \log\log k + kl)$ primes. Then, with probability at least $\frac{1}{2}$, $2^{kl} P\left((-1)^{b_1} r_1, \ldots, (-1)^{b_k} r_k\right) \bmod q$ is not 0 whenever $2^{kl} P\left((-1)^{b_1} r_1, \ldots, (-1)^{b_k} r_k\right) \neq 0$. By the Prime Number Theorem, computing the smallest $2(1 + \log n + \frac{k \log k}{2} + k \log\log k + kl)$ primes takes only $O\left((\log n \log a + \log^2 a \log\log a)^{\frac{3}{2}} \cdot (\log\log n + \log\log a)^3\right)$ time. We record these primes on a tape. Then we can generate q using at most $\log\log n + 2\log\log a + \log\log\log a + O(1)$ random bits.

Now consider the computation of $2^{kl} P\left((-1)^{b_1} r_1, \ldots, (-1)^{b_k} r_k\right) \bmod q$. For each element $u \in S \cup T$, let $e(u)$ be the number of 0's in the standard k-bit representation of u. Let $h(u)$ be the value of the monomial $f(u)$ at the points $(-1)^{b_1} 2^l r_1$, $\ldots$, $(-1)^{b_k} 2^l r_k$. Then, $2^{kl} P\left((-1)^{b_1} r_1, \ldots, (-1)^{b_k} r_k\right) = \sum_{i=1}^{n} 2^{e(s_i) l} h(s_i) - \sum_{i=1}^{n} 2^{e(t_i) l} h(t_i)$. The numbers $e(u)$ with $u \in S \cup T$ can be obtained from k in $O(n \log a)$ time by counting the number of 1's in the given binary representation of u. We do not compute the values $h(u)$ directly. Instead, we first compute $2^l r_1 \bmod q$, $\ldots$, $2^l r_k \bmod q$, and store them in the random access memory. This takes $O\left(\log a \cdot (\frac{\log n + \log a \log\log a}{\log\log n + \log\log a})^2\right)$ time. We can then obtain the values $h(u) \bmod q$ in $O(n \log a)$ time. We also compute the values $2^{e(u) l} \bmod q$ in $O\left(n(\log\log n + \log\log a)\right)$ time. We record all the values $h(u) \bmod q$ and $2^{e(u) l} \bmod q$ on a tape. Then, we can compute $2^{kl} P\left((-1)^{b_1} r_1, \ldots, (-1)^{b_k} r_k\right)$ in $O(n)$ time.

The overall error probability of Algorithm 3 is at most $\frac{3}{4}$. Summarizing the above discussion, we obtain the following theorem:

Theorem 4.2 Algorithm 2 takes $O(n \log a + \frac{\log^2 n \cdot \log a + \log^3 a \cdot (\log\log a)^2}{\log\log n + \log\log a})$ time, uses $\log a + 2\log\log n + 4\log\log a + 2\log\log\log a + O(1)$ random bits, and achieves an error probability at most $\frac{3}{4}$. Moreover, it can be modified to take $O(n \log a + (\log^2 n \cdot \log a + \log^3 a \cdot (\log\log a)^2) \cdot (\log\log n + \log\log a)^3)$ time using $\log a + \log\log n + 2\log\log a + \log\log\log a + O(1)$ random bits with an error probability at most $\frac{3}{4}$.

The error probability of Algorithm 2 can be decreased by using a larger l at step 2, or selecting more random integers at step 3, or both. This slightly increases the number of random bits and the running time. With such straightforward adjustment, Algorithm 2 improves one of Blum and Kannan's algorithms and is faster than the other for $a < 2^{\sqrt{n}/\log n}$.

5 Efficiency issues of the generic algorithm

We next discuss what conditions guarantee that the generic algorithm is efficient. Let n be the length of the given succinct representation of $Q(x_1, \ldots, x_m)$. Hereafter, by *polynomial time*, we mean time polynomial in n. We propose the following conditions on the parameters m, c, d, d_1, $\ldots$, d_m of the generic algorithm.

1. The integer m is bounded from above by a polynomial in n and can be computed in polynomial time.
2. There is a polynomial-time computable upper bound c' on c that is bounded from above by a singly exponential function in n.
3. There is a polynomial-time algorithm that computes a set of upper bounds d', d'_1, $\ldots$, d'_m on d, d_1, $\ldots$, d_m, respectively, such that d' is bounded from above by a polynomial in n and is no less than the maximum among d'_1, $\ldots$, d'_m.
4. Given a b-bit number q_i assigned to each x_i, we can evaluate $Q(q_1, \ldots, q_m)$ in time polynomial in n and b.

Note that the determinant of the Tutte matrix of a graph satisfies these conditions. The polynomial P in §4 also does.

Theorem 5.1 Assume that conditions 1 through 4 are satisfied. Suppose that the generic algorithm is modified to first compute the upper bounds c', d', d'_1, $\ldots$, d'_m and then proceed as before except that the original parameters c, d, d_1, $\ldots$, d_m are replaced by these upper bounds. Then, this modified version of the generic algorithm runs in polynomial time.

Proof. By conditions 1 and 3, s can be computed in polynomial time and is bounded from above by a polynomial in n. Hence, by the Prime Number Theorem, we can compute the smallest s primes in polynomial time in a straightforward manner. Let t be a positive integer bounded from above by a polynomial in n. By conditions 1 through 3, the number $l = t \cdot (\lceil \log(c + cd) \rceil + d\lceil \log m \rceil + 2d\lceil \log s \rceil) + d + 1$ can be computed in polynomial time and is bounded from above by a polynomial in n. Thus, we can use Newton's method to compute the square root of each $p_{i,j}$ up to the l-th bit after the decimal point in polynomial time. Since s and l are both polynomial in n, each $r'_{i,j}$ has a bit-length polynomial in n. By condition 3, each $k_i = O(\log n)$. Thus, $\sum_{j=1}^{k_1} r'_{1,j}$, $\ldots$, $\sum_{j=1}^{k_m} r'_{m,j}$ each have a bit-length polynomial in n. Then, by condition 4, $Q(\sum_{j=1}^{k_1} r'_{1,j}, \ldots, \sum_{j=1}^{k_m} r'_{m,j})$ can be computed in polynomial time. Therefore, the modified version of the generic algorithm takes polynomial time. ∎

Remark. We can also blow up the rational numbers $r_{i,j}$ to integers, and then compute the value of Q at these integers modulo a reasonably small random integer. This may greatly decrease the running time.

6 Further applications

Here, we briefly mention two other problems to which our methodology applies. One problem is to test whether three given n by n matrices A, B, and C satisfy $AB = C$. This problem was first considered by Freivalds [5] who suggested a randomized algorithm of complexity $O(n^2)$, but it required n random bits. In 1990, Noar and Noar [10] proposed a randomized algorithm of complexity $O(n^2)$, and it required $3 \log n$ random bits. Later in 1993, Kimbrel and Sinha came up with a randomized algorithm of complexity $O(n^2)$, and it required $\lceil \log n \rceil + 1$ random bits. All these algorithms achieves a probability of error at most $1/2$, and their probability of error can be decreased by using additional random bits. We can reduce the problem to the problem of polynomial testing. Then, our methodology applies and the resulting randomized algorithm runs in $O(n^2)$ time and uses $\lceil \log n \rceil$ random bits. The probability of error can be made inverse polynomially small while maintaining the same number of random bits.

The other is the problem of testing equivalence of ordered Boolean decision diagrams [2]. An *ordered Boolean decision diagram* (OBDD) of n variables x_1, ..., x_n is a labeled directed acyclic graph $D = (V, A)$ with two distinguished nodes called *start* and *finish*. There is no arc entering *start* and no arc leaving *finish*. Although any number of arcs may converge to a node, exactly zero or two arcs may exit a node. Moreover, if two arcs exit a node, one must be labeled with a variable and the other with its complement. Furthermore, at most one occurrence of every variable, complemented or not, appears on any path from *start* to *finish*.

The Boolean function represented by D is the Boolean sum of product terms in which every *product term* is a product of the labels of the edges on a path from *start* to *finish*. Two OBDD's are *equivalent* iff the Boolean functions they represent are equivalent. Blum et al. gave a randomized polynomial-time algorithm for testing the equivalence of two given OBDD's [2]. Their algorithm uses $n(\lceil \log t + \log n \rceil)$ random bits and achieves a probability of error at most $\frac{1}{t}$ for any positive integer t. We can reduce the problem to the problem of polynomial testing. Then, our methodology applies and the resulting randomized algorithm uses only n random bits to achieve a probability of error at most $\frac{1}{2}$. The probability of error can be decreased without using a single additional random bits.

References

[1] C. Berge. *Graphs and Hypergraphs.* North-Holland, New York, 1973, pp. 142-143.

[2] M. Blum, A. K. Chandra, and M. N. Wegman. Equivalence of Free Boolean Graphs Can Be Decided Probabilistically in Polynomial Time. *Information Processing Letters*, 10:80–82, 1980.

[3] M. Blum and S. Kannan. Designing Programs that Check Their Work. *J. ACM*, 42 (1995), 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. Computing*, 24 (1995), 1036-1050.

[5] R. Freivalds. Fast Probabilistic algorithms. *Lecture Notes in Computer Science 74: Proceedings of Symposium on Mathematical Foundations of Computer Science*, pages 57–69. Springer-Verlag, New York, NY, 1979.

[6] T. Kimbrel and R. K. Sinha. A Probabilistic Algorithm for Verifying Matrix Products Using $O(n^2)$ time and $\log n + O(1)$ random bits. *Information Processing Letters*, 45:107–110, 1993.

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

[8] L. Lovasz. *On Determinants, Matchings and Random Algorithms.* In L. Budach, editor, *Fundamentals of Computing Theory.* Akademia-Verlag, Berlin, 1979.

[9] N. Nisan and D. Zuckerman. Randomness is Linear in Space. *JCSS*, 52 (1996), 43-52.

[10] J. Noar and M. Noar. Small-Bias Probability Spaces: Effcient Constructions and Applications. *SIAM Journal on Computing*, 22(4):838–856, August 1993.

[11] V. Pan. Complexity of Parallel Matrix Computations. *TCS*, 54 (1987), 65-85.

[12] J. H. Reif and S. R. Tate. Optimal size integer division circuits. *SIAM Journal on Computing*, 19(5):912–924, October 1990.

[13] J. T. Schwartz. Fast Probabilistic Algorithms for Verification of Polynomial Identities. *J. ACM*, 27 (1980), 701-717.

[14] W. Thrash. A Note on the Least Common Multiples of Dense Sets of Integers. Technical Report #93-02-04, Department of Computer Science, University of Washington, Feb. 1993.

[15] W. T. Tutte. The Factors of Graphs. *Canadian Journal of Mathematics*, 4 (1952), 314-328.

[16] R. E. Zippel. Probabilistic Algorithms for Sparse Polynomials. In *Proceedings of EUROSAM 79.* Lecture Notes in Computer Science, vol. 72. Springer-Verlag, 1979, pp. 216-226.