

— 6 —

Verifying Identities, and Some Complexity

The events of September 8 prompted Foch to draft the later legendary signal: “My centre is giving way, my right is in retreat, situation excellent. I attack.” It was probably never sent.

John Keegan, The first world war

6.1. Verifying equality

6.1.1. Vectors

You are given two binary vectors $\mathbf{v} = (v_1, \dots, v_n)$, $\mathbf{u} = (u_1, \dots, u_n) \in \{0, 1\}^n$ and you would like to decide if they are equal or not. Unfortunately, the only access you have to the two vectors is via a black box that enables you to compute the dot product of two binary vectors over $\mathbb{Z}_2$. Formally, given two binary vectors as above, their dot product is $\langle \mathbf{v}, \mathbf{u} \rangle = \sum_{i=1}^n v_i u_i$ (which is a non-negative integer). Their dot product modulo 2, is $\langle \mathbf{v}, \mathbf{u} \rangle \bmod 2$ (i.e., it is 1 if $\langle \mathbf{v}, \mathbf{u} \rangle$ is odd and 0 otherwise).

Naturally, we could use the black box to read the vectors (using $2n$ calls), but since we are interested only in deciding if they are equal or not, this should require fewer calls to the black box (which is expensive).

Lemma 6.1.1. *Given two binary vectors $\mathbf{v}, \mathbf{u} \in \{0, 1\}^n$, a randomized algorithm can, using two dot-product computations modulo 2, decide if $\mathbf{v}$ is equal to $\mathbf{u}$ or not. The algorithm may return one of the following two values:*

$\neq$: Then $\mathbf{v} \neq \mathbf{u}$.

$=$: Then the probability that the algorithm made a mistake (i.e., the vectors are different) is at most $1/2$.

The running time of the algorithm is $O(n + B(n))$, where $B(n)$ is the time to compute a single dot-product of vectors of length n .

Proof: Pick a random vector $\mathbf{r} = (r_1, \dots, r_n) \in \{0, 1\}^n$ by picking each coordinate independently with probability $1/2$. Compute the two dot-products $\langle \mathbf{v}, \mathbf{r} \rangle$ and $\langle \mathbf{u}, \mathbf{r} \rangle$.

(A) If $\langle \mathbf{v}, \mathbf{r} \rangle \equiv \langle \mathbf{u}, \mathbf{r} \rangle \pmod{2} \Rightarrow$ the algorithm returns ‘=’.

(B) If $\langle \mathbf{v}, \mathbf{r} \rangle \not\equiv \langle \mathbf{u}, \mathbf{r} \rangle \pmod{2} \Rightarrow$ the algorithm returns ‘≠’.

Clearly, if ‘≠’ is returned, then $\mathbf{v} \neq \mathbf{u}$.

So, assume that the algorithm returned ‘=’ but $\mathbf{v} \neq \mathbf{u}$. For the sake of simplicity of exposition, assume that they differ on the n th bit: $u_n \neq v_n$. Since both are binary bits, we have $v_n - u_n \equiv 1 \pmod{2}$. We then have that

$$\alpha = \langle \mathbf{v}, \mathbf{r} \rangle = \overbrace{\sum_{i=1}^{n-1} v_i r_i}^{=\alpha'} + v_n r_n$$

$$\text{and } \beta = \langle \mathbf{u}, \mathbf{r} \rangle = \overbrace{\sum_{i=1}^{n-1} u_i r_i}^{=\beta'} + u_n r_n.$$

Thus, $\alpha - \beta \equiv (\alpha' - \beta') + r_n \pmod{2}$. Now, there are two possibilities:

(A) If $\alpha' \not\equiv \beta' \pmod{2}$, then, with probability half, we have $r_n = 0$, and as such $\alpha \not\equiv \beta \pmod{2}$.

(B) If $\alpha' \equiv \beta' \pmod{2}$, then, with probability half, we have $r_n = 1$, and as such $\alpha \not\equiv \beta \pmod{2}$.

As such, with probability at most half, the algorithm would fail to discover that the two vectors are different. 

6.1.1.1. Amplification. Of course, this is not a satisfying algorithm – it returns the correct answer only with probability at least half if the vectors are different. So, let us run the algorithm t times. Let $T_1, \dots, T_t$ be the returned values from all these executions. If any of the t executions returns that the vectors are different, then we know that they are different.

$$\begin{aligned} \mathbb{P}[\text{Algorithm fails}] &= \mathbb{P}[\mathbf{v} \neq \mathbf{u}, \text{ but all } t \text{ executions return ‘=’}] \\ &= \mathbb{P}[(T_1 = \text{‘=’}) \cap (T_2 = \text{‘=’}) \cap \dots \cap (T_t = \text{‘=’})] \\ &= \mathbb{P}[T_1 = \text{‘=’}] \mathbb{P}[T_2 = \text{‘=’}] \dots \mathbb{P}[T_t = \text{‘=’}] \end{aligned}$$

$$\leq \prod_{i=1}^t \frac{1}{2} = \frac{1}{2^t}.$$

We thus get the following result.

Lemma 6.1.2. *Given two binary vectors $\mathbf{v}, \mathbf{u} \in \{0,1\}^n$ and a **confidence** parameter $\delta > 0$, a randomized algorithm can decide if $\mathbf{v}$ is equal to $\mathbf{u}$ or not. More precisely, the algorithm may return one of the two following results:*

$\neq$: Then $\mathbf{v} \neq \mathbf{u}$.

$=$: Then, with probability $\geq 1 - \delta$, we have $\mathbf{v} = \mathbf{u}$.

The running time of the algorithm is $O((n + B(n)) \ln \delta^{-1})$, where $B(n)$ is the time to compute a single dot-product of two vectors of length n .

Proof: Follows from the above by setting $t = \lceil \lg(1/\delta) \rceil$. 

6.1.2. Matrices

Given three binary matrices B, C, D of size $n \times n$, we are interested in deciding if $BC = D$. Computing BC is expensive – the fastest known (theoretical!) algorithm has running time (roughly) $O(n^{2.37})$. On the other hand, multiplying such a matrix with a vector $\mathbf{r}$ (modulo 2, as usual) takes only $O(n^2)$ time (and this algorithm is simple).

Lemma 6.1.3. *Given three binary matrices $B, C, D \in \{0,1\}^{n \times n}$ and a confidence parameter $\delta > 0$, a randomized algorithm can decide if $BC = D$ or not. More precisely the algorithm can return one of the following two results:*

$\neq$: Then $BC \neq D$.

$=$: Then $BC = D$ with probability $\geq 1 - \delta$.

The running time of the algorithm is $O(n^2 \log \delta^{-1})$.

Proof: Compute a random vector $\mathbf{r} = (r_1, \dots, r_n)$, and compute the quantity $\mathbf{x} = B\mathbf{C}\mathbf{r} = B(\mathbf{C}\mathbf{r})$ in $O(n^2)$ time, using the associative property of matrix multiplication. Similarly, compute $\mathbf{y} = D\mathbf{r}$. Now, if $\mathbf{x} \neq \mathbf{y}$ then return ' $\neq$ '.

Now, we execute this algorithm $t = \lceil \lg \delta^{-1} \rceil$ times. If all of these independent runs return that the matrices are equal then return '='.

The algorithm fails only if $BC \neq D$, but then, assume the i th rows in the two matrices BC and D are different. The probability that the algorithm would not detect that these rows are different is at most $1/2$, by [Lemma 6.1.1](#). As such, the probability that all t runs failed is at most $1/2^t \leq \delta$, as desired. 

6.1.3. Checking identity for polynomials

6.1.3.1. The Schwartz–Zippel lemma Let $\mathbb{F}$ be a field (e.g., the real numbers or a finite field $\mathbb{Z}_p$). Let $\mathbb{F}[x_1, \dots, x_n]$ denote the set of polynomials over the n variables $x_1, \dots, x_n$ over $\mathbb{F}$. Such a polynomial is a sum of monomials, where a **monomial** has the form $c \cdot x_1^{i_1} \cdot x_2^{i_2} \cdots x_n^{i_n}$, where $c \in \mathbb{F}$. The **degree** of this monomial is $\text{degree}(c \cdot x_1^{i_1} \cdot x_2^{i_2} \cdots x_n^{i_n}) = i_1 + i_2 + \cdots + i_n$. Thus, a polynomial of degree d over n variables has potentially up to dn^d monomials (the exact bound is messier, but an easy lower bound on this quantity is $\binom{n}{d}$).

For a polynomial $f \in \mathbb{F}[x_1, \dots, x_n]$, the **zero set** of f , is the set

$$Z_f = \{(x_1, \dots, x_n) \mid f(x_1, \dots, x_n) = 0\}.$$

Intuitively, the zero set $Z_f = \mathbb{F}^n$ only if $f(x_1, \dots, x_n) = 0$ (and then it is the **zero** polynomial), but otherwise (i.e., $f \neq 0$) Z_f should be much “smaller”.

Specifically, a polynomial in a single variable of degree d is either zero everywhere, or has at most d roots (i.e., $|Z_f| \leq d$). This is known as the *fundamental theorem of algebra*^{6.1}. The picture gets much messier once one deals with multi-variate polynomials, but fortunately there is a simple and elegant lemma that bounds the number of zeros if we pick the values from the right set of values.

Lemma 6.1.4 (Schwartz–Zippel). *Let $f \in \mathbb{F}[x_1, \dots, x_n]$ be a non-zero polynomial of total degree $d \geq 0$, over a field $\mathbb{F}$. Let $S \subseteq \mathbb{F}$ be a finite, non-empty set. Let $\mathbf{r} = (r_1, \dots, r_n)$ be randomly and uniformly chosen from S^n . Then*

$$\mathbb{P}[f(\mathbf{r}) = 0] \leq \frac{d}{|S|}.$$

^{6.1}Wikipedia notes that the proof is not algebraic, and it is definitely not fundamental to modern algebra. So maybe it should be cited as “the theorem formerly known as the fundamental theorem of algebra”.

Equivalently, we have $|Z_f \cap S^n| \leq d|S|^{n-1}$.

Proof: The proof is by induction on n . For $n = 1$, by the theorem formerly known as the fundamental theorem of algebra, $|Z_f \cap S| \leq |Z_f| \leq d$. So assume the lemma holds for $n - 1$ variables. Since f is non-zero, it can be written as a polynomial in x_1 of degree at most d whose coefficients are polynomials in $x_2, \dots, x_n$. That is, f can be written as

$$f(x_1, \dots, x_n) = \sum_{i=0}^d x_1^i f_i(x_2, \dots, x_n),$$

where $\deg(f_i) \leq d - i$. Since f is not zero, one of the f_i s must be non-zero. Let ξ be the maximum index such that $f_\xi \neq 0$.

Now, we randomly choose the values $r_2, \dots, r_n \in S$ (independently and uniformly), and consider the polynomial in the single variable x :

$$g(x) = \sum_{i=0}^d f_i(r_2, \dots, r_n) x^i.$$

Let $\mathcal{F}$ be the event that $f_\xi(r_2, \dots, r_n) = 0$. Let $\mathcal{G}$ be the event that $g(r_1) = 0$. By induction, we have $\mathbb{P}[\mathcal{F}] \leq (d - \xi)/|S|$. If $\mathcal{F}$ does not happen, then $f_\xi(r_2, \dots, r_n) \neq 0$ and $f_i = 0$ for all $i > \xi$, so $\deg(g) = \xi$. As such, by the base case, we have that

$$\mathbb{P}[\mathcal{G} \mid \overline{\mathcal{F}}] = \mathbb{P}[g(r_1) = 0 \mid \overline{\mathcal{F}}] \leq \frac{\xi}{|S|}.$$

We conclude that

$$\mathbb{P}[f(\mathbf{r}) = 0] = \mathbb{P}[\mathcal{G} \cap \mathcal{F}] + \mathbb{P}[\mathcal{G} \cap \overline{\mathcal{F}}] \leq \mathbb{P}[\mathcal{F}] + \mathbb{P}[\mathcal{G} \mid \overline{\mathcal{F}}] \leq \frac{d - \xi}{|S|} + \frac{\xi}{|S|} \leq \frac{d}{|S|}.$$



Remark 6.1.5. Consider the polynomial $f(x, y) = (x - 1)^2 + (y - 1)^2 - 1$. The zero set of this polynomial is the unit circle centered at $(1, 1)$. So the zero set Z_f is infinite in this case. However, note that for any choice of S , the set S^2 is a grid. The Schwartz–Zippel lemma tells us that there are relatively few grid points in the zero set.

6.1.3.2. Applications.

6.1.3.2.1. Checking if a polynomial is the zero polynomial. Let f be a polynomial of degree d , with n variables, over the reals that can be evaluated in $O(T)$ time. One can check if f is zero, by picking randomly n numbers from $S = \llbracket d^3 \rrbracket$. By [Lemma 6.1.4](#), we have that the probability that f evaluates to zero on the chosen values is $\leq d/d^3 = 1/d^2$, which is a small number. As above, we can do amplification to get a high confidence answer.

6.1.3.2.2. Checking if two polynomials are equal. Given two polynomials f and g , one can now check if they are equal by checking if $f(r) = g(r)$, for some random input r . The proof of correctness follows from the above, as one interprets the algorithm as checking if $f - g$ is the zero polynomial.

6.1.3.2.3. Verifying polynomial product. Given three polynomials f, g , and h , one can now check if $fg = h$. Again, one randomly picks a value r , and checks if $f(r)g(r) = h(r)$. The proof of correctness follows from the above, as one interprets the algorithm as checking if $fg - h$ is the zero polynomial^{6.1}.

6.1.4. Checking if a bipartite graph has a perfect matching

Let $G = (L \cup R, E)$ be a bipartite graph. Let $L = \{u_1, \dots, u_n\}$ and $R = \{v_1, \dots, v_n\}$. Consider the set of variables

$$\mathcal{V} = \{x_{i,j} \mid u_i v_j \in E\}.$$

Let M be an $n \times n$ matrix, where $M[i, j] = 0$ if $u_i v_j \notin E$, and $M[i, j] = x_{i,j}$ otherwise. Let Π be the set of all permutations of $\llbracket n \rrbracket$.

A **perfect matching** is a permutation $\pi : \llbracket n \rrbracket \rightarrow \llbracket n \rrbracket$, such that for all i , we have $u_i v_{\pi(i)} \in E$. For such a permutation π , consider the monomial

$$f_\pi = \prod_{i=1}^n M[i, \pi(i)].$$

^{6.1}Some nitty-picky: Note that $\text{degree}(fg - h) \leq 2d$ when f and g have degree d , so the sample set S should be chosen of size $|S| \geq (2d)^3$ to retain the same error bound.

It is either a polynomial of degree exactly n , or it is zero. Furthermore, observe that for any two different permutations $\pi, \sigma \in \Pi$, we have that if f_π and f_σ are both non-zero, then $f_\pi \neq f_\sigma$ and $f_\pi \neq -f_\sigma$.

Consider the following “crazy” polynomial over the set of variables $\mathcal{V}$:

$$\psi = \psi(\mathcal{V}) = \det(M) = \sum_{\pi \in \Pi} \text{sign}(\pi) f_\pi,$$

where $\text{sign}(\pi)$ is the *sign* of the permutation (it is either -1 or $+1$ —for our purpose here we do not care about the exact definition of this quantity).

If there is a perfect matching in G , then there is a permutation π such that $f_\pi \neq 0$. But this implies that $\psi \neq 0$ (since it has a non-zero monomial, and the monomials cannot cancel each other).

In the other direction, if there is no perfect matching in G , then $f_\pi = 0$ for all permutations π . This implies that $\psi = 0$. Thus, deciding if G has a perfect matching is equivalent to deciding if $\psi \neq 0$. The polynomial ψ is defined via a determinant of a matrix that has variables as some of the entries (and zeros otherwise). By the above, all we need to do is to evaluate ψ over some random values. If we use exact arithmetic, we would just pick a random number in $[0, 1]$ for each variable, and evaluate ψ for these values of the variable. Namely, we filled the matrix M with values (so it is all numbers now), and we need to compute its determinant. Via Gaussian elimination, the determinant can be computed in cubic time. Thus, we can evaluate ψ in cubic time, which implies that with high probability we can check if G has a perfect matching.

If we do not want to be prisoners of the impreciseness of floating point arithmetic, then one can perform the above calculations over some finite field (usually, the field is simply working modulo a prime number).

Remark 6.1.6. Over a finite field $\mathbb{Z}_p$ with prime $p > 2n$, the Schwartz–Zippel lemma guarantees that if a perfect matching exists,

$$\mathbb{P}[\det(M) = 0] \leq n/p < 1/2,$$

yielding an efficient randomized algorithm without floating-point precision issues.

6.2. Las Vegas and Monte Carlo algorithms

Definition 6.2.1. A *Las Vegas algorithm* is a randomized algorithm that *always* returns the correct result. The only variation is that its running time might change between executions.

An example of a Las Vegas algorithm is the *QuickSort* algorithm.

Definition 6.2.2. A *Monte Carlo algorithm* is a randomized algorithm that might output an incorrect result. However, the probability of error can be diminished by repeated executions of the algorithm.

The matrix multiplication algorithm is an example of a Monte Carlo algorithm.

6.3. Bibliographical notes

The randomized algorithm for verifying matrix multiplication is due to Freivalds [Fre77]. The polynomial identity testing lemma was discovered independently by Schwartz [Sch80] and Zippel [Zip79], and earlier in a slightly weaker form by DeMillo and Lipton [DL78]. The algebraic formulation of bipartite matching via determinants of symbolic matrices is due to Edmonds [Edm67] and Tutte [Tut47]; the randomized algorithm using polynomial identity testing was introduced by Lovász [Lov79].

The term *Las Vegas algorithm* was coined by Babai [Bab79], contrasting with *Monte Carlo algorithms* (named earlier by Metropolis and Ulam).

References

- [Bab79] L. Babai. *Monte-Carlo methods in graph isomorphism*. Tech. rep. CRMA-871. Département de mathématiques et de statistique, Université de Montréal, 1979, pp. 34–50.
- [DL78] R. A. DeMillo and R. J. Lipton. *A probabilistic remark on algebraic program testing*. *IPL*, 7(4): 193–195, 1978.
- [Edm67] J. Edmonds. *Systems of distinct representatives and linear algebra*. *J. Res. Nat. Bur. Standards Sect. B*, 71B(4): 241–245, 1967.

- [Fre77] R. Freivalds. Probabilistic machines can use less running time. *Information Processing 77 (Proc. IFIP Congress 77)*, 839–842, 1977.
- [Lov79] L. Lovász. On determinants, matchings, and random algorithms. *Fundamentals of Computation Theory (FCT '79)*, 565–574, 1979.
- [Sch80] J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *JACM*, 27(4): 701–717, 1980.
- [Tut47] W. T. Tutte. The factorization of linear graphs. *J. London Math. Soc.*, s1-22(2): 107–111, 1947.
- [Zip79] R. Zippel. Probabilistic algorithms for sparse polynomials. *Symbolic and Algebraic Computation (EUROSAM '79)*, vol. 72. 216–226, 1979.