CS271 Randomness & Computation **Spring 2020**

Lecture 2: January 23

Instructor: Alistair Sinclair

Disclaimer: *These notes have not been subjected to the usual scrutiny accorded to formal publications. They may be distributed outside this class only with the permission of the Instructor.*

2.1 Testing Polynomial Identities

Randomized algorithms can be dramatically more efficient than their best known deterministic counterparts. An example of a real-world problem for which there exists a simple and efficient randomized algorithm, but for which there is no known polynomial time deterministic algorithm is that of testing polynomial identities.

The problem takes as input two polynomials Q and R over n variables, with coefficients in some field, and decides whether $Q \equiv R$. For example, if we had $Q(x_1, x_2) = (1+x_1)(1+x_2)$ and $R(x_1, x_2) = 1+x_1+x_2+x_1x_2$, the algorithm should output "Yes." This problem arises in many contexts, and is a key primitive in computer algebra packages such as Maple and Mathematica.

An obvious way to attack this problem would be to expand both Q and R as sums of monomials and compare coefficients. (Thus, for example, $Q(x_1, x_2)$ above would be expanded to $1 + x_1 + x_2 + x_1x_2$, while $R(x_1, x_2)$ is already in the correct form.) However, this method in general requires time exponential in the size of the representation of the input polynomials: consider, e.g., the polynomial $\prod_{i=1}^{n}(x_i + x_{i+1})$, which has length $O(n)$ but expands into $O(2^n)$ monomials. On the other hand, we may assume that the given polynomials can be *evaluated* efficiently at any given point $(x_1, \ldots, x_n)$. We will make use of this fact to design an efficient randomized algorithm for the problem.

2.1.1 The Schwartz-Zippel Algorithm

This algorithm, due independently to Schwartz [S79] and Zippel [Z79], is a Monte Carlo algorithm with a bounded probability of false positives and no false negatives. For simplicity, we think of the algorithm as receiving a single input polynomial $P(x_1, \ldots, x_n)$ and testing whether $P \equiv 0$. (Writing $P = Q - R$, this of course, is equivalent to testing $Q \equiv R$.) The idea of the algorithm is very simple: assign values $r_1, \ldots, r_n$ chosen independently and uniformly at random from a finite set S to $x_1, \ldots, x_n$. Then, using an efficient algorithm for evaluating P, test if $P(r_1, \ldots, r_n) = 0$, outputting "Yes" if so and "No" otherwise.

Obviously, if the algorithm outputs "No" then certainly $P \not\equiv 0$. However, if the algorithm outputs "Yes" then it is possible that in fact $P \not\equiv 0$ but $(r_1, \ldots, r_n)$ happens to be a zero of P. We now bound the probability of such an error occurring.

Claim 2.1 *If $P \neq 0$, then $\Pr[P(r_1, \ldots, r_n) = 0] \leq d/|S|$ where d is the degree of P.*

Proof: The proof is by induction on n. For the base case, $n = 1$, P is a polynomial in one variable and thus has at most d roots. Hence $\Pr[P(r_1) = 0] \leq d/|S|$.

For the inductive step, we let k be the largest degree of x_1 in P and write

$$P(x_1, \ldots, x_n) = M(x_2, \ldots, x_n)x_1^k + N(x_1, \ldots, x_n),$$

where the degree of $M(x_2, \ldots, x_n)$ is at most $d - k$ and the degree of x_1 in N is strictly less than k. For the purposes of analysis, we can assume that $r_2, \ldots, r_n$ are chosen first. Then, we let $\mathcal{E}$ be the event that $M(r_2, \ldots, r_n) = 0$. There are two cases:

- *Case 1: $\mathcal{E}$ happens.* From the induction hypothesis applied to M (a polynomial in $n-1$ variables), we know that $\Pr[\mathcal{E}] \leq (d-k)/|S|$.
- *Case 2: $\mathcal{E}$ does not happen.* In this case, we let P' be the polynomial in one variable x_1 that remains after $x_2 = r_2, \ldots, x_n = r_n$ are substituted in $P(x_1, \ldots, x_n)$. Since $M(r_2, \ldots, r_n) \neq 0$, the coefficient of x_1^k is non-zero, so P' is a non-zero polynomial of degree k in one variable. It thus has at most k roots, so $\Pr[P'(r_1) = P(r_1, \ldots, r_n) = 0 | \neg\mathcal{E}] \leq k/|S|$.

Putting the two cases together, we have

$$\begin{aligned}\Pr[P(r_1, \ldots, r_n) = 0] &= \Pr[P(r_1, \ldots, r_n) = 0|\mathcal{E}]\Pr[\mathcal{E}] + \Pr[P(r_1, \ldots, r_n) = 0|\neg\mathcal{E}]\Pr[\neg\mathcal{E}] \\ &\leq (d-k)/|S| + k/|S| = d/|S|.\end{aligned}$$

■

Thus, if we take the set S to have cardinality at least twice the degree of our polynomial, we can bound the probability of error by 1/2. This can be reduced to any desired small number by repeated trials, as usual.

Remarks

1. The Schwartz-Zippel algorithm works fine in finite fields, provided only that we can satisfy the condition $|S| > d$. Thus in particular the degree d of the polynomial being tested must be less than the size of the field.
2. A paper by Gonnet [G84] generalizes the above technique to obtain efficient randomized identity checking algorithms for a much wider range of functions, including trigonometric and logarithmic functions.
3. Is there an efficient *deterministic* algorithm for identity testing? Some efforts in this direction can be found in [LV98], and a sequence of papers by Amir Shpilka and co-authors give deterministic algorithms for various restricted classes of polynomials, such as those described by non-commutative arithmetic formulas, or by depth-3 arithmetic circuits [RS05,KS08,SV08]. On the other hand, a rather devastating negative result was proved by Kabanets and Impagliazzo [KI03], who showed that if there exists a deterministic polynomial time algorithm for checking polynomial identities, then either:

   - NEXP does not have polynomial size circuits; or
   - the Permanent cannot be computed by polynomial-size arithmetic circuits.

   This means that an efficient derandomization of the above Schwartz-Zippel algorithm (or indeed, any efficient deterministic algorithm for identity testing) would necessarily entail a major breakthrough in complexity theory. (The point here is that, although both of the above conclusions are generally believed to hold, proving either one of them would be a big deal.)

2.1.2 Application to Bipartite Matching

We begin with a nonstandard application of the Schwartz-Zippel algorithm to bipartite matching. Although bipartite matching is easy to solve in deterministic polynomial time using flow techniques, it remains an important open question whether there exists an efficient *parallel* deterministic algorithm for this problem.

Bipartite matching is the following problem: Given a bipartite graph $G = (V_1, V_2, E)$, where $|V_1| = |V_2| = n$, and all edges connect vertices in V_1 to vertices in V_2, does G contain a perfect matching, that is, a subset of exactly n edges such that each vertex is contained in exactly one edge?

To obtain an efficient randomized algorithm for this problem, we begin with a definition:

Definition 2.2 (Tutte matrix) *The Tutte matrix A_G corresponding to the graph G is the $n \times n$ matrix $[a_{ij}]$ such that a_{ij} is a variable x_{ij} if $(i, j) \in E$, and 0 otherwise.*

Claim 2.3 *G contains a perfect matching if and only if* $\det(A_G) \neq 0$.

Proof: By definition, $\det(A_G) = \sum_\sigma \operatorname{sgn}(\sigma) \prod_{i=1}^n a_{i\sigma(i)}$, where the sum is over all permutations σ of $\{1, \ldots, n\}$. Note that each monomial in this sum corresponds to a *possible* perfect matching in G, and that the monomial will be non-zero if and only if the corresponding matching is present in G. Moreover, every pair of monomials differs in at least one (actually, at least two) variables, so there can be no cancellations between monomials. This implies that $\det(A_G) \not\equiv 0$ iff G contains a perfect matching. ■

The above Claim immediately yields an efficient algorithm for testing whether G contains a perfect matching: Simply run the Schwartz-Zippel algorithm on $\det(A_G)$, which is a polynomial in n^2 variables of degree n. Note that the determinant can be computed in $O(n^3)$ time by Gaussian elimination. Moreover, the algorithm can be efficiently parallelized using the standard fact that an $n \times n$ determinant can be computed in $O(\log^2 n)$ time on $O(n^{3.5})$ processors [Ber84].

Exercise: Generalize the above to a non-bipartite graph G. For this, you will need the skew-symmetric matrix $A_G = [a_{ij}]$ defined as:

$$a_{ij} = \begin{cases} x_{ij} & \text{if } (i, j) \in E \text{ and } i < j; \\ -x_{ij} & \text{if } (i, j) \in E \text{ and } i > j; \\ 0 & \text{otherwise.} \end{cases}$$

Hint: You should show that the above Claim still holds, with this modified definition of A_G. This requires a bit more care than in the bipartite case, because the monomials in $\det(A_G)$ do not necessarily correspond to perfect matchings, but to *cycle covers* in G. In this case some cancellations will occur.

2.1.3 Fast Parallel Algorithms for Finding a Perfect Matching

We now turn to the problem of actually finding a perfect matching. One way to do this would be to reuse the previous algorithm in the following way (assuming we have previously tested G and discovered that it almost certainly has a perfect matching):

```
Input: $G = (V, E)$ and $M = \emptyset$
while $M$ is not a perfect matching
    pick an arbitrary edge $(i, j)$ of $G$
    test if $G' = G \setminus \{i, j\}$ has a perfect matching using the Schwartz-Zippel algorithm
    if Schwartz-Zippel outputs "Yes"
        then let $M = M \cup \{(i, j\})$ and $G = G'$
        else let $G = G \setminus \{(i, j)\}$
    end if
end while
output $M$
```

Since the size of E decreases by 1 at each iteration of the while loop, after at most $|E|$ stages we will have constructed a matching. (A detail here is that, since the Schwartz-Zippel test is being invoked several times, we need to reduce its error probability on each invocation to less than $1/|E|$ using repeated trials, so that the overall error probability remains bounded.) Unfortunately, this approach is inherently sequential and does not yield an efficient parallel algorithm. Instead, we present a randomized algorithm due to Mulmuley, Vazirani and Vazirani [MVV87] that is parallelizable.

We first prove a key lemma; this version is slightly stronger than the original and is due to Noam Ta-Shma.

Lemma 2.4 [Isolation Lemma] *Let $S_1, \ldots, S_k$ be subsets of a set S of cardinality m. Let each element $x \in S$ have a weight w_x, chosen independently and uniformly at random from the set $\{1, \ldots, \ell\}$. Then*

$$\Pr[\exists \text{ a unique set } S_i \text{ of minimum weight}] \geq \left(1 - \frac{1}{\ell}\right)^m \geq 1 - \frac{m}{\ell}.$$

(Here the weight *of set S_i is defined as $\sum_{x \in S_i} w_x$.)*

Proof: First note that we may assume w.l.o.g. that no set S_i is a subset of any other set S_j; for if so then S_j cannot have minimum weight and so may be safely ignored.

Now let $\mathcal{W}$ denote the set of all weight functions $w = \{w_x\}$, and $\mathcal{W}^+$ the subset of $\mathcal{W}$ satisfying $w_x > 1$ for all $x \in S$. We claim that to each function $w \in \mathcal{W}^+$, we can associate a *distinct* function $w' \in \mathcal{W}$ such that w' has a unique minimum weight set. This will complete the proof since then

$$\Pr[\exists \text{ a unique set } S_i \text{ of minimum weight}] \geq \frac{|\mathcal{W}^+|}{|\mathcal{W}|} = \left(1 - \frac{1}{\ell}\right)^m \geq 1 - \frac{m}{\ell}.$$

To verify the above claim, given a function $w \in \mathcal{W}^+$, pick an arbitrary subset S_i of minimum weight under w and define

$$w'_x = \begin{cases} w_x - 1 & \text{if } x \in S_i\,; \\ w_x & \text{otherwise.} \end{cases}$$

Then clearly S_i is the unique minimum weight set under w'. And moreover, we can recover w from w' simply by increasing the weights of all elements in S_i by 1, so the w's are distinct. ■

Note that if we take $\ell = 2m$ (as we will in our application below) then we get a unique minimum weight set with probability at least 1/2.

Armed with the Isolation Lemma, we can give a parallel algorithm to find perfect matchings in bipartite graphs. The set S in the Isolation Lemma will simply be the edge set E of the graph, and $S_1, \ldots, S_k$ are the perfect matchings in G. We pick a weight w_{ij} for each edge (i, j) uniformly at random from the set $\{1, \ldots, 2|E|\}$. The weight $w(M)$ of a matching M is the sum of the weights of the edges in M. We then form the matrix B by replacing x_{ij} in the Tutte matrix by $2^{w_{ij}}$.

The role of the Isolation Lemma is to "break symmetry" between the perfect matchings in G. The uniqueness of the minimum weight matching means that all processors are able to work towards the same goal in parallel, without explicit coordination.

Here is the rest of the algorithm, which either outputs a perfect matching M or fails:

```
calculate $2^w$, the largest power of 2 that divides $det(B)$
**for** each edge $(i,j)$ in parallel **do**
    compute $t_{ij} = 2^{w_{ij}} \det(B_{ij})$, where $B_{ij}$ is the $(i,j)$ minor of $B$
    place $(i,j)$ in $M$ iff the largest power of 2 that divides $t_{ij}$ is $2^w$
**if** $M$ is a perfect matching **then** output $M$ **else** output *fail*.
```

We note that this algorithm can be implemented to run in polylog time on polynomially many processors, since this applies to each determinant computation and the t_{ij} can be computed in parallel. The next Claim shows that, if there is a unique minimum weight matching M, then the algorithm always outputs it. Together with the Isolation Lemma, this implies that the algorithm succeeds with probability at least 1/2. Note that the algorithm ends by checking that it has indeed found a perfect matching, because if the minimum weight matching is not unique we have no control over what the algorithm does. This ensures that any non-fail output is always correct.

Claim 2.5 *If the minimum weight perfect matching M is unique, the above algorithm outputs it.*

Proof: Let M_0 be the unique minimum weight perfect matching. Then the value w computed in the first line of the algorithm is just the weight of M_0; this follows from the fact that

$$\det(B) = \sum_{M \in \mathcal{M}(G)} \pm 2^{w(M)},$$

where $\mathcal{M}(G)$ is the set of perfect matchings in G, and M_0 is unique, so $\det(B)/2^{w(M_0)}$ is odd.

Moreover, for any edge $(i,j) \in E$, $\det(B_{ij})$ corresponds to the perfect matchings in $G \setminus \{i,j\}$, i.e.,

$$\begin{aligned}\det(B_{ij}) &= \sum_{M \in \mathcal{M}(G \setminus \{i,j\})} \pm 2^{w(M)} \\ &= 2^{-w(i,j)} \sum_{M \cup (i,j) \in \mathcal{M}(G)} 2^{w(M \cup (i,j))}.\end{aligned}$$

This implies that the largest power of 2 dividing the quantity t_{ij} computed by the algorithm will still be 2^w iff (i,j) is an edge of M_0. Hence the algorithm correctly identifies M_0 assuming that it is unique. ∎

Exercise: Generalize this algorithm to nonbipartite graphs.

The question of whether a perfect matching can be found *deterministically* in polylog time on polynomially many processors remains one of the central open problems in parallel algorithms. Recently, major progress has been made on this question by Fenner, Gurjar and Thierauf [FGT16] and by Svensson and Tarnawski [ST17]. Both papers achieve polylog time but require *quasi*-polynomially many (i.e., $n^{\text{polylog}\, n}$) processors. The key ingredient in both papers is a derandomization of the Isolation Lemma in the particular context where the S_i are perfect matchings in a graph. [FGT16] does this for bipartite graphs, and [ST17] extends it (non-trivially) to general graphs. Even more recently, Anari and Vazirani [AV18] gave a polylog time algorithm with polynomially many processors for the special case of *planar* graphs, using quite different techniques.

References

[AV18] N. Anari and V.V. Vazirani. "Planar graph matching is in NC." *Proceedings of the 59th IEEE Symposium on Foundations of Computer Science*, 2018, pp. 650–661.

[Ber84] S.J. Berkowitz. "On computing the determinant in small parallel time using a small number of processors." *Information Processing Letters* **18** (1984), pp. 147–150.

[FGT16] S.A. Fenner, R. Gurjar and T. Thierauf. "Bipartite perfect matching is in quasi-NC." *Proceedings of the 48th ACM Symposium on Theory of Computing*, 2016, pp. 754–763.

[G84] G.H. Gonnet, "Determining equivalence of expressions in random polynomial time," *Proceedings of the 16th ACM Symposium on Theory of Computing*, 1984, pp. 334–341.

[KI03] V. Kabanets and R. Impagliazzo, "Derandomizing polynomial identity tests means proving circuit lower bounds," *Proceedings of the 35th ACM Symposium on Theory of Computing*, 2003, pp. 355–364.

[KS08] Z.S. Karnin and A. Shpilka, "Black Box Polynomial Identity Testing of Generalized Depth-3 Arithmetic Circuits with Bounded Top Fan-In," *Proceedings of the IEEE Conference on Computational Complexity* 2008, pp. 280–291.

[LV98] D. Lewin and S. Vadhan, "Checking Polynomial Identities," *Proceedings of the 39th IEEE Symposium on Foundations of Computer Science*, 1998, pp. 1–6.

[MVV87] K. Mulmuley, U.V. Vazirani and V.V. Vazirani, "Matching is as easy as matrix inversion," *Combinatorica* **7** (1987), pp. 105–113.

[RS05] R. Raz and A. Shpilka, "Deterministic polynomial identity testing in non-commutative models," *Computational Complexity* **14** (2005), pp. 1–19.

[S80] J.T. Schwartz, "Fast probabilistic algorithms for verification of polynomial identities," *Journal of the ACM* **27** (1980), pp. 701–717.

[SV08] A. Shpilka and I. Volkovich, "Read-once polynomial identity testing," *Proceedings of the ACM STOC* 2008, pp. 507–516.

[ST17] O. Svensson and J. Tarnawski. "The matching problem in general graphs is in quasi-NC." *Proceedings of the 58th IEEE Symposium on Foundations of Computer Science*, 2017, pp. 696–707.

[Z79] R.E. Zippel, "Probabilistic algorithms for sparse polynomials," *Proceedings of EUROSAM* 1979, Springer Lecture Notes in Computer Science Vol. 72, pp. 216–226.