

## Lecture 16: November 5

Lecturer: Micah Adler

Scribe: Shengda Wang, Ying Gong

This lecture includes the following:

- **Application of Schwartz's Algorithms in Perfect Matching Problem.**
- **Primality Testing.**

## 16.1 Application of Schwartz's Algorithms

In this section we will use Schwartz's Algorithm to solve the perfect matching problem. Consider a bipartite graph  $G = (U, V, E)$  with the independent sets of vertices  $U = \{u_1, u_2, \dots, u_n\}$  and  $V = \{v_1, \dots, v_n\}$ . A *matching* is a collection of edges  $M \subseteq E$  such that each vertex occurs at most once in  $M$ . A *perfect matching* is a matching of size  $n$ .

### 16.1.1 Perfect Matching Problem

**Input:** A bipartite graph  $G = (U, V, E)$ , where  $|U| = |V| = n$ .

**Question:** Does  $G$  contain a perfect matching?

### 16.1.2 Network Flow Solution

This problem can be solved by Network Flow algorithm. Using Even-Tarjan [ET75] algorithm it can be solved in time  $O(m\sqrt{n})$ .

### 16.1.3 Solve the Problem By Schwartz's Algorithm

We can use Schwartz's Algorithm to solve the perfect matching problem.

Let  $M[G]$  be the  $n \times n$  adjacency matrix for  $G = (U, V, E)$  defined as follows:

$$M[G](i, j) = \begin{cases} 1 & \text{if } (u_i, v_j) \in E \\ 0 & \text{otherwise} \end{cases}$$

Let  $A[G]$  be an  $n \times n$  matrix derived from  $G$  as follows:

$$A[G](i, j) = \begin{cases} x_{ij} & \text{if } M[G](i, j) = 1 \\ 0 & \text{otherwise} \end{cases}$$

**Example:**

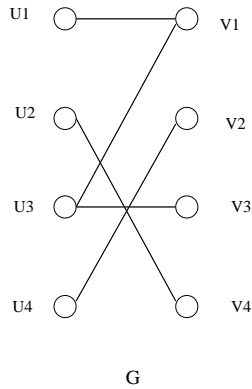


Figure 16.1: A Bipartite Graph

For  $G = (U, V, E)$ , where  $|U| = |V| = 4$ :

$E = \{(u_1, v_1), (u_3, v_1), (u_2, v_4), (u_3, v_3), (u_4, v_2)\}$ , so:

$$A[G] = \begin{bmatrix} x_{11} & 0 & 0 & 0 \\ 0 & 0 & 0 & x_{24} \\ x_{31} & 0 & x_{33} & 0 \\ 0 & x_{42} & 0 & 0 \end{bmatrix}$$

Before prove Edmonds' Theorem, we introduce some definitions.

**Definition 16.1** A permutation  $\sigma$  of the set of integers  $\{1, 2, \dots, n\}$  is an arrangement of these integers in some order without omissions or repetitions.

**Example:** There are six different permutations of the set of integers  $\{1, 2, 3\}$ . These are:

$$(1, 2, 3), (2, 1, 3), (3, 1, 2), (1, 3, 2), (2, 3, 1), (3, 2, 1)$$

**Definition 16.2** Let  $c(\sigma)$  be the number of cycles in  $\sigma$ ,

$$\text{sgn}(\sigma) = \begin{cases} 1 & \text{if } c(\sigma) \equiv n \pmod{2} \\ -1 & \text{if } c(\sigma) \not\equiv n \pmod{2} \end{cases}$$

The number of cycles in  $\sigma$  can be found by writing  $\sigma$  in cyclic notation: a cycle begins with some integer  $i$  from the set that is not yet in another cycle. To complete the cycle, compute  $\sigma(i), \sigma(\sigma(i)), \dots, \sigma^k(i)$  until  $\sigma^k(i) = i$ . Repeat until no integers remain.

**Example:** For the permutation  $\sigma = (2, 3, 1, 5, 4, 6)$ , shown below, we have  $\sigma_{cyclic} = (1, 2, 3)(4, 5)(6)$ ,  $c(\sigma) = 3$ , and  $\text{sgn}(\sigma) = -1$ .

$i$	$\sigma(i)$
1	2
2	3
3	1
4	5
5	4
6	6

**Definition 16.3** Let  $A$  be a square matrix. The determinant of  $A$ ,  $\det(A)$  as follows:

$$\det(A) = \sum_{\sigma \in S} \left( \text{sgn}(\sigma) \prod_{i=1}^n A(i, \sigma(i)) \right)$$

where  $S$  is the set of permutations of  $\{1, 2, \dots, n\}$ .

Example: For a  $3 \times 3$  matrix

$$M = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

and

$$S = \{(1, 2, 3), (1, 3, 2), (2, 1, 3), (2, 3, 1), (3, 1, 2), (3, 2, 1)\}$$

we have:

$$\det(M) = aei - afh - bdi + bfg + cdh - ceg$$

**Theorem 16.4 (Edmonds' Theorem)** A bipartite graph  $G$  contains a perfect matching if and only if  $\det(A[G]) \neq 0$ .

**Proof:**

Let  $A[G]$  be the  $n \times n$  matrix derived from a bipartite graph  $G = (U, V, E)$ , where  $|U| = |V| = n$ . Since each variable  $x_{ij}$  occurs at most once in  $A[G]$ , there can not be any cancellation of items in the summation. Therefore the determinant is not identically zero if and only if there is a permutation  $\sigma$  for which the corresponding term in the summation is non-zero. The latter happens if and only if each of the entries  $A[G](i, \sigma(i))$ ,  $1 \leq i \leq n$ , is non-zero. This is equivalent to having a perfect matching (corresponding to  $\sigma$ ) in  $G$ .

**So:**

no perfect matching  $\Rightarrow$  every term  $\equiv 0$ ;

perfect matching  $\Rightarrow$  non-zero terms  $\Rightarrow \det(A[G]) \neq 0$  (because no terms can cancel).

■

We can determine the existence of perfect matching by testing whether  $\det(A[G]) \equiv 0$ . One straightforward way to test whether  $\det(A[G]) \equiv 0$  is to calculate  $\det(A[G])$  symbolically. The problem of this method is that the number of permutations is exponential, that is, computing the determinant is actually the same as searching every possible perfect matching. Here we will introduce a much faster approach using Schwartz's algorithm to determine whether or not  $\det(A[G]) \equiv 0$ .

**Algorithm:**

**Input:**  $G = (U, V, E)$

Let  $M = A[G]$

Let  $S = \{0, \pm 1, \pm 2, \dots, \pm n\}$ , where  $n$  is the degree of the polynomial

$\forall i, j$  s.t.  $x_{ij}$  appears in  $M$ , set  $x_{ij} =$  some random element in  $S$ .

Evaluate  $\det(M)$ ,

**if**  $\det(M) = 0$ , output "no perfect matching"

**otherwise** output "perfect matching"

If there is no perfect matching then  $Pr(\text{correct}) = 1$ . If there is a perfect matching, then  $Pr(\text{correct}) = 1 - \frac{n}{2n+1} > \frac{1}{2}$ . We can keep repeating the runs and each run is independent, so that eventually we can achieve an arbitrarily small probability of error.

**Run time** of evaluating  $\det(M)$ .

We said earlier that computing  $\det(M)$  symbolically is exponential. When substituting numbers in the matrix there turns out to be a much faster way. Roughly, the running time is the same as that of matrix multiplication. That is, suppose  $\mu(n)$  is the speed of matrix multiplication, then the running time is:  $O(\mu(n)) = O(n^{2.37})$ .

For a non-random (deterministic) algorithm, the running time is:  $O(m\sqrt{n})$ . When the graph is dense, it can be  $O(n^{2.5})$ .

Therefore, asymptotically speaking, the random algorithm is faster than the non-random algorithm. Its advantages in practice are:

- It is simple and straightforward. There are plenty of numerical packages that will compute the determinant of a matrix.
- There is an easy way to extend the algorithm from a bipartite graph to an arbitrary graph; using a non-random algorithm to deal with an arbitrary graph is much more complicated and slower.

## 16.2 Primality Testing

In this section, we will see more applications of random algorithms. One of them is primality testing.

**Input:**  $n$ .

**Question:** Is  $n$  prime?

### 16.2.1 Algorithm 1

Test every integer in  $[2, \dots, \sqrt{n}]$  to see if it divides  $n$ .

**Problem:** The running time is exponentially big, because the input size is  $\lceil \log n \rceil$ .

### 16.2.2 Algorithm 2

Before we discuss the second algorithm, it is necessary to introduce a theorem as the basis of Algorithm 2. It comes from number theory.

**Theorem 16.5 (Fermat's Little Theorem)** *If  $n$  is prime then  $\forall a \in [1, n), a^{n-1} \equiv 1 \pmod{n}$ .*

**Algorithm 2:**

Choose  $a$  uniformly from  $[2, n-1]$  at random.

if  $a^{n-1} \not\equiv 1 \pmod{n}$ , then  $n$  is composite.

otherwise  $n$  is prime.

**Run time:** Polynomial because  $a^{n-1} \bmod n$  can be computed in time  $O(\log^c n)$  for some constant  $c$ . This is left as an exercise.

**Problem:** (Carmichael numbers)

There exist some specific composites called *Carmichael numbers*, such that  $a^{n-1} \equiv 1 \pmod{n} \forall a \in [1, n)$ , and the algorithm does not return the correct answer. Carmichael numbers are rare. The first 5 Carmichael numbers: 561, 1105, 1729, 2465, and 2821. These "bad inputs" do exist no matter how many times we do the algorithm. Let us find another better algorithm that is based on the Miller-Rabin test in the following section.

### 16.2.3 Algorithm 3

**Miller-Rabin test:** [CLR90, Chapter 33]

If  $n$  is composite then for at least half of the  $a \in (1, n)$ , either  $a^{n-1} \not\equiv 1 \pmod{n}$  or  $\exists k$ , s.t.  $r = \frac{n-1}{2^k}$  is an integer and  $1 < \gcd(a^r - 1, n) < n$ .

**Running time:** Polynomial, because checking if  $a^{n-1} \equiv 1 \pmod{n}$  is polynomial, and we can try different values of  $k$  for at most  $\log n$  times, and the gcd part is also polynomial. (Euclid invented an algorithm for gcd with the running time of  $O((\log n)^2)$  in 300 B.C.)

**Algorithm 3:**

```
repeat
  Pick  $a \in (1, n)$  uniformly at random.
  if the Miller-Rabin test says  $n$  is composite, return "composite" and stop
until the chance of a composite surviving is negligible.
return "prime."
```

After repeating  $t$  times,

If  $n$  is prime: Probability[return "prime,"]=1

If  $n$  is composite: Probability[return "composite"] $\geq 1-2^{-t}$

### 16.2.4 Algorithm 4

In addition to the Miller-Rabin test, there is another test which also runs in polynomial time. It is the **Adelman-Huang test** (not described here). It is sort of opposite to Miller-Rabin test in terms of performance, that is:

If  $n$  is composite: Probability[ returns "composite" ]=1

If  $n$  is prime: Probability[returns "composite"] $\leq \frac{1}{2}$ .

## References

A81 H. ANTON, *Elementary Linear Algebra*, 1981, pp. 57–63.

CLR90 T. CORMEN, and C. LEISERSON and R. RIVEST, *Introduction to Algorithms*, MIT Press, 1990.

