CS761 Derandomization and Pseudorandomness **2022-23 Sem I**

Lecture 20: 20-10-22

Scribe: Punna Hitesh Kumar *Lecturer: Rohit Gurjar*

In next few lectures, we will see a randomized algorithm for the bipartite matching problem and a partial derandomization for it.

Bipartite Matching

Definition 20.1. *Bipartite graph (or bigraph) is a graph whose vertices can be divided into two disjoint and independent sets U and V, that is every edge connects a vertex in U to one in V. Vertex sets U and V are usually called the left and the right parts of the vertices.*

Definition 20.2. *Let $G = (V, E)$ be a graph. A matching in G is a set of edges $M \subseteq E$ such that for every $e, e' \in M$, there is no vertex v such that e and e' are both incident on v.*
The matching M is called perfect if for every $v \in V$, there is some $e \in M$ which is incident on v.

We are going to look at some algebraic algorithms for solving the bipartite matching problem (given by Lovász, and Mulmuley, Vazirani, Vazirani) using the determinant of the bi-adjacency matrix. The bi-adjacency matrix $A(G)$ for a bipartite graph G is a matrix where the rows correspond to the vertices in the left part, the columns correspond to the vertices in the right part, and the (i, j) entry is 1 if there is an edge between ith left vertex and the jth right vertex, 0 otherwise.

Definition 20.3. *For a matrix $M \in R^{n\times n}$, its determinant can be expressed as*

$$\det(M) = \sum_{\sigma\in S_n} sign(\sigma)\Pi_{i=1}^{n} M_{i,\sigma(i)}$$

Here, $sign(\sigma) = -1^{swaps(\sigma)}$, where $swaps(\sigma)$ = number of swaps to generate the permutation σ.

Observe that for the bi-adjacency matrix $A(G)$ of a bipartite graph G, the term $\Pi_{i=1}^{n} A(G)_{i,\sigma(i)}$ is nonzero if and only if it corresponds to a perfect matching, that is, for each i the edge $(i, \sigma(i))$ is present in the graph. Hence, the determinant can be written as

$$det(A(G)) = \sum_{M\in \text{ Perfect Matchings in } G} sign(M) \cdot 1$$

The following claim is immediate.

Claim 20.4. *If G does not have a perfect matching then* $\det(A(G)) = 0$.

However, the other direction is not true. $\det(A(G)) = 0$ can be zero even when G contains a perfect matching. To make the other direction work as well, the idea is to replace the 1 entry in the bi-adjacency matrix with random integers (from a small range).

Claim 20.5. *Let us replace each nonzero entry in $A(G)$ with a random value (chosen uniformly and independently) from $\{1, 2, \dots, 2n\}$. If G has a perfect matching then $det(A(G)) \neq 0$ with probability at least 1/2.*

It turns out that the above claim can be proved using Schwartz-Zippel lemma. View each nonzero entry in $A(G)$ as a variable. Then the determinant is a polynomial of degree-n. Now we can use the below lemma for proving the above claim.

Lemma 20.6. *Let $\boldsymbol{P}$ be a nonzero polynomial in m variables and degree d. Then*

$$Pr_{\alpha\in\{1,2,3,\dots,2d\}^m}[P(\alpha) \neq 0] \geq 1/2.$$

The lemma can be easily proved for a univariate polynomial. We can use induction to extend it to a larger number of variables.

Thus, we get a randomized algorithm to decide if a given graph has a perfect matching. Interestingly, this algorithm is also interesting from the parallelization perspective, because determinant can be computed very efficiently (poly-logarithmic time) in parallel. The next question to ask is if there is also a parallel algorithm for the construction version of the problem: given a graph, find a perfect matching if one exists. Given the decision algorithm, we can use the standard self-reduction to find a perfect matching. Choose an edge e, remove it and check if the graph still has a perfect matching. If yes, then continue with the obtained subgraph after removing e. Otherwise include the edge e into perfect matching and recurse on the subgraph obtained after removing endpoints of e. Clearly, this algorithm is inherently sequential and not suitable for parallelization.

Can we design a parallel algorithm (with poly-logarithmic time) to find a perfect matching? Let's consider a special case when the given graph G has exactly one perfect matching. To identify this perfect matching we could do the following: for each edge e in parallel, remove endpoints of e and check if there is a perfect matching in the graph. If yes, then e is part of the unique perfect matching. Clearly, this approach will not work for the case where we have more than one perfect matching because the final edges set could just be the set of all edges, which does not give us anything useful.

Lemma 20.7. *Mulmuley, Vazirani, Vazirani 1987 Suppose we can assign (small) weights on the edges $w : E \to Z$ such that the minimum weight perfect matching is unique (here $w(M) := \sum_{e \in M} w(e)$). Then we can find that minimum weight perfect matching in using an efficient parallel algorithm.*

Algorithm. Construct the adjacency matrix with non-zero entries replaced with $2^{w(e)}$. Note the term in the determinant corresponding to a matching M will be $sign(M)2^{w(M)}$.

- Compute the determinant and find the maximum power of 2 that divides the determinant. This value will be the weight of the unique minimum weight perfect matching.
- For each edge e in parallel: decrease the weight of e by 1 compute the determinant again. If the minimum weight also decreases by 1 then e is part of the minimum weight perfect matching.

The only question remains is how to assign weights to ensure a unique minimum weight perfect matching. The idea of Mulmuley, Vazirani, Vazirani was to assign weights randomly independently. Here, we need to careful about the range of the weights. If the weights are too large, we might not be able to compute the determinant efficiently. On the other hand, if the weights are too small, they might not ensure a unique minimum perfect matching.

Lemma 20.8 (Isolation Lemma)**.** *Let us assign each edge a random weight from $\{1, 2, \ldots, 2m\}$ independently. Then the minimum weight perfect matching is unique with probability at least 1/2.*