CS273 Foundations of Parallel and Distributed Systems **Spring 2003**

Lecture 12: March 5

Lecturer: Satish Rao *Scribe: Jeffrey Pang*

12.1 Matching

12.1.1 Review

During the last lecture we began to describe a randomized NC (RNC) algorithm for finding a matching in a bipartite graph (as a special case of the max-flow problem, which is generally P-complete). The algorithm starts by representing a bipartite graph G as a matrix A of *indeterminants* x_{ij}s. Each row of A corresponds to a vertex u_i on the left-hand side of the graph, and each column refers to a vertex on the right-hand v_j side. Specifically,

$$a_{ij} = \begin{cases} x_{ij} & \text{if } (u_i, v_j) \text{ is an edge in } G \\ 0 & \text{otherwise} \end{cases} \tag{12.1}$$

For example, the bipartite graph in Figure 12.1 would be represented as follows:

$$A_{example} = \begin{bmatrix} x_{11} & x_{12} & 0 \\ x_{21} & 0 & 0 \\ 0 & x_{32} & x_{33} \end{bmatrix} \tag{12.2}$$

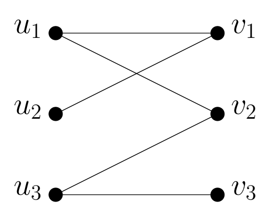

Figure 12.1: Example bipartite graph $G_{example}$.

We argued that there exists a matching in G if and only if $det(A) \not\equiv 0$. To see this, notice that:

$$\det(A) = \sum_{\pi} (-1)^{\sigma(\pi)} a_{1\pi(1)} a_{2\pi(2)} a_{3\pi(3)} \cdots \tag{12.3}$$

In other words, it is the sum of products of each permutation of the a_{ij}s such that the first element comes from the first row, the second comes from the second row, etc. But any permutation of this form that is non-zero implies a matching in G (since each indeterminant represents compatibility between a vertex on the left and a vertex on the right). Moreover, each of these permutations is unique, so the sum of any of them (or their negation) that are non-zero will yield a non-zero result.

Nevertheless, there are an exponential number of terms in the determinant, so we can not efficiently multiply it out to get the coefficients.

12.1.2 Randomized Polynomial Evaluation

To address this problem, we note that $\det(A)$ is a polynomial of degree n. We can randomly assign values to each indeterminant x_{ij}, calculate the determinant of the resulting matrix, and then see if the result is 0 or not.

Theorem 12.1 *Given a polynomial $p(x_1, x_2, ..., x_n)$ with max degree d, if we pick $r_1, r_2, ..., r_n$ uniformly at random from $\{1, 2, ..., 2d\}$, then $\Pr[p(x_1, ..., x_n) \not\equiv 0 \wedge p(r_1, ..., r_n) = 0] \leq \frac{1}{2}$.*

The proof was omitted, but it is easily seen by induction on the number of variables n. Thus, if we pick values for each x_{ij} from $\{1, ..., 2n^2\}$, if $\det(A) \not\equiv 0$ then it will evaluate to non-zero with constant probability.

Another issue is how we compute these determinants in parallel. The solution is to use Chistov's algorithm. We did not go into details.

Finally, although we can use this algorithm to determine if a matching exists, it does not tell us what that matching was.

12.1.3 Finding the Matching

Before we show how to find the matching, we present the following important lemma:

Lemma 12.2 Isolation Lemma. *Given a family $\mathcal{F}$ of subsets of $\{1, 2, ..., n\}$, by randomly choosing element weights $w(i)$ from $\{1, 2, ..., 2n\}$, the weight of the smallest weight subset is unique with probability $\frac{1}{2}$.*[1]

This lemma is somewhat surprising because there are only n^2 different values that subset weights can take on, while there can be an exponential number of subsets. We present the following proof of Lemma 12.2 by Joel Spencer:

Proof: Define the function α as:

$$\alpha(x) = \min_{E \in \mathcal{F}: x \notin E} w(E) - \min_{E \in \mathcal{F}: x \in E} w(E - \{x\}) \tag{12.4}$$

for elements x, subsets E, and weights $w(i)$. Notice that $\alpha(x)$ is independent of $w(x)$, so we can calculate it without knowing the weight of x. Therefore, we can treat $w(x)$ as a random variable uniformly distributed over $\{1, ..., 2n\}$.

There are $2n$ different weights each element can take on, so:

$$\Pr[w(x) = \alpha(x)] \leq \frac{1}{2n} \Rightarrow \tag{12.5}$$

$$\Pr[w(x) = \alpha(x) \text{ for some } x] \leq \frac{1}{2} \tag{12.6}$$

Now suppose $A, B \in \mathcal{F}$ are both minimal sets, and $y \in A - B$. Then:

$$\min_{E \in \mathcal{F}: y \notin E} w(E) = w(B) \tag{12.7}$$

$$\min_{E \in \mathcal{F}: y \in E} w(E - \{y\}) = w(A) - w(y) = w(B) - w(y) \tag{12.8}$$

$$\Rightarrow$$

$$\alpha(y) = w(B) - (w(B) - w(y)) = w(y) \tag{12.9}$$

[1] We define the weight of the subset E to be $w(E) = \sum_{x \in E} w(x)$.

We are done, since this occurs with probability $\leq \frac{1}{2}$. ■

Note that since the proof just uses a union bound to bound the probability (see equation (12.6)), we can make the bound arbitrarily small by choosing weights from a larger set of numbers. For example, in the following analysis we choose weights from $\{1, ..., 2n^2\}$, so the probability of a distinct subset with minimal weight is $1 - \frac{1}{2n}$.

Now, to find a matching we first choose w_{ij}s each at random from $\{1, ..., 2n^4\}$ (the number of indeterminants in A is $\leq n^2$). Let $B = A$ with each x_{ij} replaced with $2^{w_{ij}}$.[2] By Lemma 12.2, there exists a unique matching of minimal weight with probability at least $1 - \frac{1}{2n^2}$ because every matching can be viewed as a subset of the edges in G (and therefore $\mathcal{F}$ is the set of matchings).

Second, we compute $\det(B)$ (using Chistov's algorithm). We see that:

$$\det(B) = \sum_{\pi} (-1)^{\sigma(\pi)} \prod_{i} 2^{w_{i\pi(i)}} \tag{12.10}$$

$$= \sum_{\pi} (-1)^{\sigma(\pi)} 2^{\sum_i w_{i\pi(i)}} \tag{12.11}$$

Notice that we can view the value of this summation as a bit string, where each summand contributes 1 bit (since they are each a power of 2). If we assume that the matching with minimal weight is unique, then the lowest order 1 in this bit-string is precisely the weight of the matching with minimal weight. This is because each summand contributing a 1 is a non-zero term in the determinant which represents a matching, and if the minimal is unique, then there can not be any term that adds a bit at or before it.[3]

Next, for each x_{ij}, we compute $\det(B_{ij})$ where B_{ij} is B with the ith row and jth column removed. If x_{ij} was in the minimal weight matching then $\det(B_{ij})$ should yield an even smaller minimal weight (lowest order 1-bit). In particular, the π term that yielded the lowest order 1-bit in $\det(B)$ should now be equal to:

$$(-1)^{\sigma(\pi)} 2^{\left(\sum_i w_{i\pi(i)}\right) - w_{ij}} \tag{12.12}$$

Thus, if the positions of lowest order 1-bits of $\det(B)$ and $\det(B_{ij})$ differ by w_{ij} then x_{ij} is in the matching with high probability. By the union bound, we find the edges in the matching with probability $\geq \frac{1}{2}$. (This can be made 1 - inverse polynomial by increasing the size of the set that we choose weights from.)

12.2 Maximal Independent Set

12.2.1 Review

Previously we showed that finding a greedy maximal independent set in a graph $G(V, E)$ is P-complete. However, this assumed that we start constructing the set from a particular node in the graph; instead we can try to find *any* max independent set.

If we were finding a max independent set sequentially, we could use the following algorithm:

1. Find some independent set X_i.

[2] One issue here is that these numbers are very big, but the size of each is still on the order of the largest possible weight. In addition, we can look at the bits in a bit-string in parallel.

[3] The number may be negative. In this case, if the bit-string represents the value in twos-complement, then the position of the lowest order 0 in the bit-string is the minimal weight.

2. Remove all nodes in X_i and neighbors of nodes in X_i.
3. Repeat until the graph is empty.

The max independent set is $X = \bigcup_i X_i$.

12.2.2 A Parallel Algorithm

This gives us some intuition for the following parallel algorithm:

1. For each node u (in parallel), put it in X with probability $\frac{1}{\text{degree}(u)}$.
2. If both u and v were chosen and (u, v) is an edge in the graph, then remove the node with smaller degree from X, breaking ties randomly.
3. Remove all nodes and edges incident to $X \cup N(X)$, where $N(X)$ is the set of neighbors to nodes in X.
4. Repeat until the graph is empty.

This is similar to the list ranking problem, where we would like to find an independent set of "lazy" elements.

Lemma 12.3 *The expected number of edges incident to $X \cup N(X)$ is $\geq \frac{1}{8}|E|$.*

This was not proven. The total work performed is:

$$W(|E|) = W(\frac{15}{16}|E|) + C_v|E| \tag{12.13}$$

where C_v is a random variable which is equal to the amount of work we are required to do to get the number of edges down to $\frac{15}{16}|E|$. In other words, we don't recurse until we reduce the number of edges by the constant factor.

By Lemma 12.3, $E[C_v|E|] \leq \frac{1}{8}|E|$. By the Markov bound, this means we can remove $\frac{1}{16}|E|$ edges at each step with constant probability. Because we get rid of a constant factor of edges at each step, we can prove the depth is $D(|E|) = O(\log|E|)$ with high probability (using a Chernoff bound).

12.3 Sorting

This section is adapted from [BM97].

12.3.1 Quick Sort

The parallel algorithm for quick sort on an array A is shown in Algorithm 1.

If we optimistically assume that the partition element p divides the set into two equal partitions, the work and depth are, respectively:

$$W(n) = 2W(n/2) + O(n) \tag{12.14}$$
$$D(n) = 2W(n/2) + 1 \tag{12.15}$$

Algorithm 1 QUICKSORT(A)

```
if |A| = 1 then return A
i ← rand_int(|A|)
p ← A[i]
in parallel do
    L ← QUICKSORT({a : a ∈ |a < p})
    E ← {a : a ∈ A|a = p}
    G ← QUICKSORT({a : a ∈ A|a > p})
return L ∘ E ∘ G
```

which have solutions $W(n) = O(n \log n)$ and $D(n) = O(\log n)$.

In practice, we can improve this by partitioning the elements into P sets, one for each processor. However, like the load balancing problem, we are unlikely to assign approximately the same number of elements to each processor. This can be improved by choosing *sp* candidate partition elements, sorting the candidates, and choosing the candidates with ranks $s, 2s, ..., (p-1)s$.

12.3.2 Radix Sort

Radix sort examines the keys to be sorted one digit at a time, starting with the least significant digit. The intermediate sort on each digit must be *stable*; that is, the output ordering must preserve the input order of any two keys with identical values in the position being examined. A parallel radix sort algorithm can be implemented using the PREFIX-SUMS (SCAN) subroutine [BM97].

It is notable that parallel sorting algorithms are also used for sorting on systems with multiple disks since they can reduce the number of I/Os required and more easily overlap computation with disk-to-memory transfers.

References

[BM97] G.E. Blelloch and B.M. Mags, "Parallel Algorithms," In A. B. Tucker, Jr., editor, *The Computer Science and Engineering Handbook*, CRC Press, Boca Raton, FL, 1997, pp. 277-315. `http://www-2.cs.cmu.edu/afs/cs/project/phrensy/pub/papers/BlellochM96b.ps`.