Lecture 5

1 Constructing Small Sample Spaces

1.1 Finding Large Cuts in a Graph

Definition 1.1.1. The VERTEX-PARTITION problem:
Given a graph $G = (V, E)$, partition $V$ into two sets $A$ and $B$ such that more than $1/2$ of the edges in $E$ cross the partition $(A, B)$.

Theorem 1.1.2. For any graph, there exists a solution to VERTEX-PARTITION.

Proof. Consider the random partition $(A, B)$ constructed by tossing a fair coin for every vertex to determine whether to place this vertex in $A$ or in $B$. Formally, for each vertex $i$, $1 \leq i \leq |V| = n$, define an independent random variable

$$X_i = \begin{cases} +1 & \text{with probability } 1/2 \\ -1 & \text{with probability } 1/2 \end{cases} \rightarrow \text{vertex } i \in A$$

An assignment to the $X_i$s specifies a point in the sample space, and corresponds to a particular partition of the vertices of $G$.

Clearly $\forall i, E[X_i] = 0$. Construct a variable $\varepsilon_{i,j}$ that has a value of 1 precisely when the edge $(i, j)$ crosses the partition $(A, B)$ i.e. $X_i \neq X_j$, and has a value of 0 otherwise. $\forall (i, j) \in E$,

$$\varepsilon_{i,j} = \frac{(1 - X_iX_j)}{2}$$

Let $S$ denote the number of edges that cross the partition $(A, B)$.
\[ E[S] = \sum_{(i,j) \in E} E[\xi_{i,j}] \]
\[ = \sum_{(i,j) \in E} E\left[\frac{1 - X_i X_j}{2}\right] \]
\[ = \frac{|E|}{2} - \frac{\sum_{(i,j) \in E} E[X_i X_j]}{2} \]
\[ = \frac{|E|}{2} - \frac{\sum_{(i,j) \in E} E[X_i] E[X_j]}{2} \]
\[ = \frac{|E|}{2} , \text{ since } E[X_i] = E[X_j] = 0. \]

Since the average number of edges that cross a randomly constructed partition is \( \frac{|E|}{2} \), there must exist a specific “good” partition which achieves this. \( \Box \)

We know that a “good” partition exists, but how can we find it? The obvious way is to exhaustively search the entire sample space\(^3\). Unfortunately, since there are \( O(2^n) \) possible assignments to the \( X_i \)'s, we cannot hope to guarantee finding the “good” partition efficiently. In the quest for a polynomial-time algorithm, we can either attempt to (i) reduce the size of the sample space, or (ii) find a “good” point more intelligently than by exhaustive search. In the next section we pursue the former of these two approaches.

### 1.2 A Smaller Sample Space for VERTEX-COVER

The proof of Theorem 1 uses the fact that \( E[X_i X_j] = E[X_i] E[X_j] \), and hence implies that \( \forall i, j \) we choose whether to place vertex \( i \) in \( A \) or \( B \) independently of where we choose to place vertex \( j \). The key observation is that in this proof we only require pairwise independence of our choices (amongst the \( X_i \)'s), whereas our current method of selecting the \( X_i \)'s (just flipping a fair coin for each \( X_i \)) gives us mutual (full) independence amongst the \( X_i \)'s.

\(^3\)Actually, it turns out that for the VERTEX-PARTITION problem a randomly constructed partition is “good” with high probability, but this is not true in general for other problems. In the worst case one may have to search the entire sample space exhaustively before finding the solution.
We now construct a smaller sample space of size $O(n)$ that will provide us with (precisely) pairwise independence amongst the $X_i$s.

Let us assume that $n = |V|$ is a power of 2. We define

$$w = w_1w_2w_3\cdots w_{\log n}$$

to be a random string of length $(\log n)$ whose $i$th bit is independently randomly chosen to be

$$w_i = \begin{cases} 1 & \text{with probability } 1/2 \\ 0 & \text{with probability } 1/2 \end{cases}$$

For each vertex $i$, $1 \leq i \leq |V| = n$ define

$$X_i = (-1)^{\text{bin}(i) \cdot w}$$

where $\text{bin}[i]$ is the binary expansion of $i$. For example,

$$X_5 = (-1)^{\text{bin}(000\cdots00101) \cdot (w_1w_2w_3\cdots w_n)} = (-1)^{(w_n - 2 + w_n)}$$

As before $X_i = 1 \rightarrow$ vertex $i \in A$, and $X_i = -1 \rightarrow$ vertex $i \in B$.

It is not hard to see that as long as $w$ is not the string of all 0s

$$Pr[X_i = -1] = Pr[X_i = 1] = \frac{1}{2}$$

$$\forall i, j \text{ s.t. } i \neq j, Pr[X_i = a|X_j = b] = \frac{1}{2}, \text{ where } a, b \in \{0, 1\}.$$ 

Thus choosing a random (not all 0s) string $w$ corresponds to making a set of pairwise independent random assignments to the $X_i$s. This is because if $i$ does not equal $j$, then the binary representation of $i$ and $j$ must differ in at least one bit. Choose a bit $m$ where they differ, and suppose $i$ has a 1 in bit $m$ and $j$ has a 0 in bit $m$ in its binary representation. Then the independent coin that sets the value of $m$ will flip or retain the value of $X_i$ based on all the other bits, completely independently at random from all the random choices that went into the value for $X_j$, since the value of $X_j$ does not depend on this coin since $j$'s $m$th bit is 0. The expected number of "good" edges over
this smaller space is exactly the same as over the larger (exponential-sized) space, since the old proof of **Theorem 1** still holds. Most importantly, this new sample space (consisting of all possible strings $w$) is only linear in size, since it has $2^{\log n} = n$ sample points$^4$. Given this smaller sample space, we can find the “good” point $w$ efficiently by simply searching all of the $n$ possible sample points.

### 2 Using Pessimistic Estimates

#### 2.1 Finding Large Independent Sets in a Graph

**Definition 2.1.1.** The Maximal Independent Set (MIS) of a graph is a set of vertices $I \subseteq V$ s.t.

$$
\begin{align*}
x \in I & \rightarrow \ \forall y \ s.t. \ (x, y) \in E, y \notin I \\
x \notin I & \rightarrow \ \exists y \ s.t. \ (x, y) \in E, \land y \in I
\end{align*}
$$

It is easy to compute MIS sequentially since we can just greedily add vertices$^5$. The question is can MIS computed efficiently in parallel? We address this question in the section that follows.

#### 2.2 A Randomized Parallel Algorithm for MIS

The algorithm$^6$ progresses in rounds. During each round, all the remaining vertices that are not already in the MIS and do not have any neighbors in the MIS, “compete” to get in. They do this as follows:

Vertex $i$ flips a biased coin whose value is the random variable

$$
X_i = \begin{cases} 
1 \text{ with probability } p & \rightarrow \text{ vertex } i \text{ tries to enter the MIS} \\
0 \text{ with probability } (1 - p) & \rightarrow \text{ vertex } i \text{ does not try to enter the MIS}
\end{cases}
$$

(The precise value of $p$ will fall out of this analysis.) If vertex $i$ tries to enter and all the neighbors of $i$ do not try to enter, we can insert the vertex $i$ into

---

$^4$In general if we require $k$-wise independence (for some constant $k$) amongst the $X_i$s, we can construct a sample space with $O(N^{\lfloor 1/2 \rfloor})$ points.

$^5$Add a vertex to the MIS, remove it and all its neighbors from the graph. Repeat.

$^6$This algorithm is based on Luby’s parallel algorithm for determining the maximal independent set of a graph.
the MIS. Formally, we define a variable $Y_i$ whose value is 1 precisely when vertex $i$ is admitted into the MIS.

$$Y_i = \begin{cases} 1 & \text{if } (X_i = 1) \cdot \prod_{(i,j) \in E} (X_j = 0) \\ 0 & \text{otherwise} \end{cases}$$

Call a vertex *satisfied* if either it enters, or some neighbor enters. Formally, we define a variable $Z_i$ whose value is 1 precisely when vertex $i$ is “satisfied”.

$$Z_i = \begin{cases} 1 & \text{if } (Y_i + \sum_{(i,j) \in E} Y_j) \geq 1 \\ 0 & \text{otherwise} \end{cases}$$

Let $D$ be the largest vertex degree found in the graph. In general $D = O(n)$. Construct $\log D$ buckets and distribute the “unsatisfied” vertices of the graph into these buckets as follows: Bucket $i$ contains those “unsatisfied” vertices with degree $\in [2^{i-1}, 2^i]$.

The underlying claim we will prove is that at the end of each round, the algorithm expects to capture a constant fraction $\frac{1}{\alpha}$ of the top bucket of biggest degree nodes.

If this claim is true, then we will satisfy all big degree vertices in $O(\log n)$ rounds, since $D \leq n$. Furthermore, since each round can be done in $O(\log n)$ time on a PRAM\(^7\), the entire algorithm can run in parallel in $O(\log^2 n)$ time.

Define $T$ to be the number of big degree vertices “satisfied” in a given round, and let $B$ be the set of big degree vertices at the start of that round.

$$T = \sum_{i \in B} Z_i$$

We want to show that

$$E[T] \geq \frac{|B|}{\alpha} \text{ for some constant } \alpha.$$  

Recall that

$$E[Y_j] = E[X_j \prod_{(j,k) \in E} (1 - X_k)]$$

\(^7\)See Appendix on the PRAM Model.
so making any statements about $E[T]$ based on the $Y_i$s requires $D$-wise independence amongst the $X_i$s. Since $D$ can be as large as $n$, this is obviously not going to lead to a small sample space, nor a polynomial algorithm.

### 2.2.1 Lower Bounding $E[T]$ via a Pessimistic Estimate

In our analysis of $E[T]$ we circumvent requiring $D$-wise independence amongst the $X_i$s by using the following technique: Rather than analyzing $T$, we find some other quantity $T'$ that is (i) an underestimate of $T$, (ii) requires only pairwise independence among the $X_i$s, and (iii) is also at least as large as a constant fraction of the “big degree” vertices. Formally, we want to find a “pessimistic estimator” or benefit function $T'$ such that

$$E[T] \geq E[T'] \geq \frac{|B|}{\alpha}$$

for some constant $\alpha$, and $T'$ relies on only pairwise independence amongst the $X_i$s.

Define new variable $R_{i,j}$ (to be used in place of the $Y_i$s), as follows:

$$R_{i,j} = X_j \prod_{(j,k) \in E} (1 - X_k) \prod_{(i,l) \in E, l \neq j} (1 - X_l)$$

In other words, $R_{i,j}$ is 1 exactly when vertex $i$ is satisfied by precisely one of its neighbors, namely vertex $j$, and by no other. Clearly $R_{i,j}$ is 1 under a restriction of the general conditions for a vertex being “satisfied”, therefore

$$Z_i \geq \sum_{(i,j) \in E} R_{i,j}$$

So if we define

$$T' = \sum_{i \in B} \sum_{(i,j) \in E} R_{i,j}$$

then

$$T' \leq \sum_{i \in B} Z_i = T$$

Therefore $T'$ is an underestimate of $T$. This gives us property (i).
Looking further at $T'$ we note that

$$T' = \sum_{i \in B} \sum_{(i,j) \in E} R_{i,j}$$

$$\geq \sum_{i \in B} \sum_{(i,j) \in E} X_j (1 - \sum_{(j,k) \in E} X_k - \sum_{(i,l) \in E} X_l)$$

Thus analyzing $E[T']$ requires only pairwise independence amongst the $X_i$s. This gives us property (ii). We further note that

$$T' = \sum_{i \in B} \sum_{(i,j) \in E} X_i (1 - \sum_{(j,k) \in E} X_k - \sum_{(i,l) \in E} X_l)$$

$$= \sum_{i \in B} \sum_{(i,j) \in E} (p - \sum_{(j,k) \in E} p^2 - \sum_{(i,l) \in E, l \neq j} p^2)$$

$$\geq |B| \frac{D}{2} (p - Dp^2 - Dp^2)$$

where $D$ is the maximum vertex degree in the graph. Setting $p = \frac{1}{4D}$ yields $\alpha = \frac{1}{16}$. This gives us property (iii).

Thus we have shown that the randomized algorithm always captures a constant fraction ($\frac{1}{16}$) of the "big degree" vertices. This immediately implies that the algorithm runs in $O(\log n)$ phases each taking $O(\log n)$ time.
3 Appendix: The PRAM Model

PRAM is an acronym for Parallel Random Access Machine. A PRAM is an abstract model of parallelism consisting of $N$ processors together with $M$ memory locations organized into $N$ blocks. In a shared memory PRAM, any processor can access any memory location in a single step. (See Figure 1.)

![Shared memory PRAMs diagram](image)

Figure 1: Shared memory PRAMs.

There are several variations of PRAMs depending on whether or not we allow concurrent reads or concurrent writes. We consider the least restrictive of these:

**CRCW** - Concurrent Read Concurrent Write. At any step, each memory can be accessed by many processors, for both writing and reading the same location. There are also several variations on what kinds of concurrent writes are allowed and on how conflicts are resolved. We choose the simplest of these: All processors writing to the same memory location must write identical values.