1 Notes by Raymond Hogenson

In this lecture, we discuss

1. Deterministic algorithm for the Lovász Local Lemma;
2. $k$-Universal sets;
3. Error correcting codes for insertions and deletions.

2 Review

2.1 Limited Independence

In lecture 9, we learned how to generate random bits with limited independence using fewer truly random bits. In fact, some randomized algorithms do not need full independence, and limited independence suffices for them. Suppose we use $\ell$ truly random bits to generate $n$ random variables with limited independence. If $\ell = O(\log n)$, then $2^\ell$ is polynomial in $n$. That means we can check every single seed from those $\ell$ truly random bits in polytime. This will derandomize your randomized algorithm where limited independence is sufficient.

Recall the following definition for $k$-wise independence.

**Definition 1** ($k$-wise independence). Let $X_1, \ldots, X_n$ be random variables, with respective ranges $A_1, \ldots, A_n$. We say that they are $k$-wise independent if for all subsets $S = \{s_1, \ldots, s_\ell\}$ of $[n]$ with $\ell \leq k$ and for all $a_i \in A_i$

$$
\Pr\left[\bigwedge_{i=1}^{\ell} X_{s_i} = a_i\right] = \prod_{i=1}^{\ell} \Pr[X_{s_i} = a_i].
$$

In lecture 9, we showed that we can generate $\mathbb{F}_q$-valued $k$-wise independent random variables $X_1, \ldots, X_n$ for some prime $q = O(n)$ using only $O(k \log n)$ truly random bits. The following shows that we can generate $n$ binary $k$-wise independent random variables $X_1, \ldots, X_n$ using $O(k \log n)$ truly random bits.

**Claim 1.** We can generate binary $k$-wise independent random variables $X_1, \ldots, X_n$ using $O(k \log n)$ bits.

**Proof.** With $O(k \log n)$ bits, we can generate $\mathbb{F}_q$-valued $k$-wise independent random variables $Y_1, \ldots, Y_n$ for some prime $q = O(n)$. Let’s define a random variable $X_i$ for each $i$ as $X_i := Y_i \pmod{2}$. It is sufficient to show that $X_1, \ldots, X_n$ are $k$-wise independent.
Note that for all $S = \{s_1, \ldots, s_\ell\} \subseteq [n]$ with $|S| = \ell \leq k$ and $b_1, \ldots, b_n \in \{0, 1\}$,
\[
\Pr \left[ \bigwedge_{i=1}^{\ell} X_{s_i} = b_i \right] = \Pr \left[ \bigwedge_{i=1}^{\ell} Y_{s_i} \equiv b_i \pmod{2} \right]
\]
\[
= \Pr \left[ \bigwedge_{i=1}^{\ell} \bigvee_{t_i \equiv b_i \pmod{2}} Y_{s_i} = t_i \right]
\]
\[
= \sum_{(t_1, \ldots, t_\ell) \equiv (b_1, \ldots, b_\ell)} \Pr \left[ \bigwedge_{i=1}^{\ell} Y_{s_i} = t_i \right]
\]
\[
= \prod_{i=1}^{\ell} \sum_{t_i \equiv b_i} \Pr[Y_{s_i} = t_i]
\]
\[
= \prod_{i=1}^{\ell} \Pr[X_{s_i} = b_i]
\]

By Claim 1, we are able to generate $\operatorname{poly}(n)$ number of binary random variables using $O(k \log n)$ bits. In the analysis of the Moser-Tardos algorithm, binary random variables are used to fill in the randomness table. We can replace them with our $k$-wise independent random variables to fill in the randomness table. Since there are $2^{O(k \log n)}$ seeds from $O(k \log n)$ bits, we need to check $2^{O(k \log n)}$ corresponding tables.

Later, we will show that we need $C \log n$-wise independence for some constant $C$ in order to derandomize the Moser-Tardos algorithm. Then we need to use $O\left((\log n)^2\right)$ random bits. However, using $O\left((\log n)^2\right)$ random bits is not appropriate for generating a deterministic algorithm since $2^{O((\log n)^2)} = n^{O(\log n)}$. Instead of using $O(\log n)$-wise independence, we can take advantage of $\delta$-approximate $O(\log n)$-wise independence. When $\delta = 1/poly(n)$, we can also get some high probability results. Note the following.

**Definition 2** ($\delta$-approximate $k$-wise independence). Let $X_1, \ldots, X_n$ be random variables, with respective ranges $A_1, \ldots, A_n$. We say that they are $\delta$-approximate $k$-wise independent if for all subsets $S = \{s_1, \ldots, s_\ell\}$ of $[n]$ with $\ell \leq k$ and for all $a_i \in A_i$
\[
\left| \Pr \left[ \bigwedge_{i=1}^{\ell} X_{s_i} = a_i \right] - \prod_{i=1}^{\ell} \Pr[X_{s_i} = a_i] \right| < \delta.
\]

**Claim 2.** We can generate binary $\delta$-approximate $k$-wise independent random variables $X_1, \ldots, X_n$ using $O(k + \log n + \log \frac{1}{\delta})$ bits.

The above claim implies that we can generate polynomially many binary $1/poly(n)$-approximate $O(\log n)$-wise independent random variables using only $O(\log n)$ bits. Since $2^{O(\log n)} = O(n^c)$ for some constant $c$, there are only polynomially many randomness tables to check to get a deterministic algorithm.

### 2.2 The Moser-Tardos Algorithm

In lecture 4 we learned the Moser-Tardos algorithm which provides a constructive proof of the Lovász Local Lemma. Here, we restate some technical details which are used in the analysis of the algorithm.
The following is setting for the symmetric version of the Lovász Local Lemma. We have \( n \) binary random variables \( X_1, \ldots, X_n \) and \( m \) events \( A_1, \ldots, A_m \). Let \( \text{vbl}(A_i) \) denote the set of random variables that \( A_i \) depends on for each \( i \). Let \( k_{\text{min}} := \min_{1 \leq i \leq m} |\text{vbl}(A_i)| \) and \( k_{\text{max}} := \max_{1 \leq i \leq m} |\text{vbl}(A_i)| \). We can assume that \( k_{\text{max}} \) is a constant, i.e., each event \( A_i \) depends on only a constant number of random variables. Then we can draw the corresponding dependency graph. Let us assume that the maximum degree of the dependency graph is \( \Delta \) and \( \Pr[A_i] = \ldots = \Pr[A_m] = p \) for some constant \( p \in [0, 1] \). By the Lovász Local Lemma, if \((\Delta + 1)p < 1\), we know that \( \Pr[A_1 \land \cdots \land A_m] > 0 \). In other words, there is an assignment of \( X_1, \ldots, X_n \) such that all of \( A_1, \ldots, A_m \) are violated.

Note the following claims.

**Claim 3.** If \((\Delta + 1)p < 1 - \epsilon\), then there is no consistent witness tree of size greater than \( \frac{\epsilon \log n}{c} \) with high probability where \( c \) is a constant.

**Proof.** Note that the probability that there is a witness tree of size at least \( K = \frac{\epsilon \log n}{c} \) is at most

\[
\sum_{i=1}^{m} \sum_{s=K}^{\infty} \Pr[\exists \text{ consistent witness tree rooted at } A_i \text{ of size } s],
\]

by the union bound. We also know that the number of consistent witness trees rooted at \( A_i \) of size \( s \) is at most \((\Delta + 1)^s\). Let \( \tau \) be a witness tree rooted at \( A_i \) of size \( s \). The probability that \( \tau \) is consistent with the table is equal to the probability that the event in each node of \( \tau \) is violated by the assignment given in the table. Since each entry in the randomness table is independent of the other entries, one node is independent from the other nodes. The probability that the event corresponding to each node is violated is \( p \). Therefore, the probability that \( \tau \) is consistent with the table is \( p^s \). Therefore, by the union bound, we get

\[
\sum_{i=1}^{m} \sum_{s=K}^{\infty} \Pr[\exists \text{ consistent WT rooted at } A_i \text{ of size } s] \leq \sum_{i=1}^{m} \sum_{s=K}^{\infty} \left( \frac{\Delta s}{s-1} \right) p^s \leq \sum_{i=1}^{m} p \sum_{s=K}^{\infty} (\Delta p)^{s-1}
\]

Note that \( \sum_{s=K}^{\infty} (\Delta p)^{s-1} \leq \frac{1}{(1-\Delta p)\Delta p} (\Delta p)^K \). In addition, we can also prove that \( \sum_{i=1}^{m} p < n/e \). The proof for that is following. For each random variable \( X_i \), consider the subgraph \( C_i \) of the dependency graph induced by the nodes where their corresponding events depend on \( X_i \). Clearly, \( C_i \) is a clique. Besides, \( C_1, \ldots, C_n \) cover all the nodes of the dependency graph. Therefore,

\[
\sum_{i=1}^{m} p(A_i) \leq \sum_{j=1}^{n} \sum_{A \in C_j} p(A) \leq \sum_{j=1}^{n} (\Delta + 1)p < n/e.
\]

since the size of each clique is at most \( \Delta + 1 \) and we assumed that \((\Delta + 1)p < 1\). Since \( K = \frac{\epsilon \log n}{c} \), we also get that

\[
\frac{1}{(1-\Delta p)\Delta p} (\Delta p)^K \leq \frac{1}{C'} (1-\epsilon)^{\epsilon \log n} = \frac{1}{C'} n^{1-\epsilon}\log(1-\epsilon) \leq \frac{1}{C''} n^{-c/2}
\]

where \( C' \) is a constant. Then, \( \sum_{i=1}^{m} p \sum_{s=K}^{\infty} (\Delta p)^{s-1} \leq \frac{1}{C''} n^{1-c/2} \). We can choose a sufficiently large \( c \) so that there is no consistent witness tree of size at least \( \frac{\epsilon \log n}{c} \) with high probability.

**Claim 4.** If \((\Delta + 1)p < 1 - \epsilon\), the total number of resamplings is \( O\left(\frac{n \log n}{\epsilon k_{\text{min}}}\right) \) with high probability.
Lemma 5. If $C$ exist consistent witness tree of size greater than $c \log n / \epsilon$. Let $r$ denote the number of total resampling steps. Since each event depends on at least $k_{\min}$ random variables, at least $k_{\min}$ variables are resampled in each resampling step. Then,

$$rk_{\min} \leq \frac{cn \log n}{\epsilon}.$$  

Hence, $r \leq \frac{n \log n}{rk_{\min}}$. \qed

3 Deterministic Algorithm for the Lovász Local Lemma

We proved in Claim 3 that there is no consistent witness tree of size greater than $C \log n$ with high probability for some constant $C$ when we use full independence. To fill in the randomness table, you need $\Omega(n^c)$ random variables for some constant $c$. However, you cannot derive an efficient deterministic algorithm out of it in this case, because there are $2^{O(n^c)}$ tables to consider.

In order to get a deterministic algorithm for the Lovász Local Lemma, we will use $\delta$-approximate $k$-wise independent binary random variables. When $\delta = 1/poly(n)$ and $k = O(\log n)$, we need only $O(\log n)$ random bits to generate polynomially many $\delta$-approximate $k$-wise independent random variables. That means there are $2^{O(\log n)} = n^{O(1)}$ seeds, so we need to check polynomially many tables to get a satisfying assignment. However, the problem is Claim 3 does not hold in this case. If a witness tree $\tau$ has size $\omega(\log n)$, then we cannot directly compute the probability of $\tau$ being consistent. In other words, we cannot apply the union bound to bound the probability that there exist consistent witness tree of size greater than $K$.

Fortunately, we have another strategy. In fact, we can prove that there is a table among those $n^{O(1)}$ tables where there is no consistent witness tree of size greater than $C \log n$ for some constant $C$. The following two lemmas show that.

Lemma 5. If $\tau$ is a consistent witness tree, then there exists a consistent witness tree of size in the interval $[|\tau| - 1, \Delta + 1]$. \newline

Proof. Note that there are at most $\Delta + 1$ child nodes of the root of $\tau$, since the root is allowed to have a child node with the same event. Let $v_1, \ldots, v_\ell$ be the child nodes of the root where $\ell \leq \Delta + 1$. Consider the subtree of $\tau$ rooted at $v_i$ for each $i$, and say it is $\tau_i$. By the Pigeonhole principle, the maximum of $|\tau_1|, \ldots, |\tau_\ell|$ is at least $\frac{|\tau| - 1}{\Delta + 1}$. Without loss of generality, we may assume that $|\tau_1|$ is the maximum. Let us generate the witness tree rooted at $v_1$. Then each node that was in $\tau_1$ will be attached to the witness tree at some point. It is possible that some nodes in $\tau_i$ for $i \neq 1$ are in the witness tree. Thus, the size of the witness tree is at least $|\tau_1|$. \qed

By Lemma 5, if there is no consistent witness tree of size in the interval $[C \log n, C(\Delta + 1) \log n]$, then there is no consistent witness tree that has size greater than $C(\Delta + 1) \log n$ for any constant $C$. Thus, there is no consistent witness tree of size greater than $C \log n$ in this case.

Lemma 6. We fill in the randomness table using $\delta$-approximate $k$-wise independent random variables where $\delta = 1/n^c$ and $k = (\Delta + 1)Ck_{\max} \log n$ for some constants $C$ and $c$. Then, there is a table such that there is no consistent witness tree of size in $[C \log n, C(\Delta + 1) \log n]$. \newline

Proof. It is sufficient to show the probability that there is a consistent witness tree of size in $[C \log n, C(\Delta + 1) \log n]$ is less than $1$. Let $\tau$ be a witness tree of size $s \in [C \log n, C(\Delta + 1) \log n]$. Note that

$$\Pr\{\tau \text{ is consistent with the randomness table}\} - p^s < \delta,$$
since we use \( \delta \)-approximate \( k \)-wise independent random variables and \( \tau \) depends on at most \( sk_{\max} \) entries of the randomness table. Then, we get the following.

\[
\sum_{i=1}^{m} \sum_{s=C \log n}^{\infty} \Pr[\exists \text{ consistent WT rooted at } A_i \text{ of size } s] \leq \sum_{i=1}^{m} \sum_{s=K}^{\infty} \left( \frac{\Delta s}{s-1} \right) (p^s + \delta).
\]

To complete the proof, we need to argue that there are constants \( C \) and \( c \) such that \( \sum_{i=1}^{m} \sum_{s=K}^{\infty} \left( \frac{\Delta s}{s-1} \right) (p^s + \delta) \) is less than 1. Firstly, consider the following.

\[
\sum_{i=1}^{m} \sum_{s=K}^{\infty} \left( \frac{\Delta s}{s-1} \right) p^s \leq \sum_{i=1}^{m} \frac{C(\Delta+1) \log n}{p \log n} \sum_{s=K}^{\infty} (\Delta e)^{s-1} \leq n \left( C \Delta (\log n) \right) (\Delta e)^{C \log n \log n} = \frac{C}{pe} n (\log n \cdot (\log p \log n)) = \frac{C}{pe} n^{1-C(1/(\Delta e)) \log n}
\]

Secondly, consider the following.

\[
\sum_{i=1}^{m} \sum_{s=K}^{\infty} \frac{\Delta s}{s-1} \delta \leq \frac{\delta}{p} \sum_{i=1}^{m} \frac{C(\Delta+1) \log n}{p \log n} \sum_{s=K}^{\infty} (\Delta e)^{s-1} \leq \frac{\delta}{p} n \left( C \Delta (\log n) \right) (\Delta e)^{C \log n \log n} = \frac{C\delta}{pe} n^{1-C(1/(\Delta e)) \log n}
\]

Therefore, we can choose both \( c \) and \( C \) such that \( \sum_{i=1}^{m} \sum_{s=K}^{\infty} \left( \frac{\Delta s}{s-1} \right) (p^s + \delta) \) is less than 1. \( \square \)

By Lemmas 5 and 6, there is a table such that there is no consistent witness tree of size greater than \( C \log n \) when we fill in the randomness table using \( (1/n^c) \)-approximate \( (C(\Delta + 1)k_{\max} \log n) \)-wise independent random variables. Then, the total number of resampling steps is at most \( Cn^{\log n} / k_{\min} \), so \( Cn^{\log n} / k_{\min} \) entries of the table are used. Hence, we can use \( (1/n^c) \)-approximate \( (C(\Delta + 1)k_{\max} \log n) \)-wise independent binary random variables \( Y_1, \ldots, Y_{Cn^i/k_{\min}} \) to fill in the randomness table. In this case, we use

\[
O\left( (C(\Delta + 1)k_{\max} \log n + \log \left( \frac{Cn^3}{k_{\min}} \right) \right) \right) = O\left( (C(\Delta + 1)k_{\max} + \frac{3C}{k_{\min}} + c) \log n \right)
\]

truly random bits, and thus there are \( n^{\theta(\Delta_{k_{\max}})} \) tables to consider. The deterministic algorithm for the Lovász Local Lemma is as follows.

**Algorithm 1:** Deterministic LLL algorithm

**Input:** \( n \) variables \( X_1, \ldots, X_n \) and \( m \) events \( A_1, \ldots, A_m \) where \( \text{vbl}(A_i) \) denotes the set of random variables that \( A_i \) depends on for each \( i \);

**Generate** all \( n^{\theta(\Delta_{k_{\max}})} \) possible tables that contain only \( Cn^2 / k_{\min} \) resampling steps by using \( (1/n^c) \)-approximate \( (C(\Delta + 1)k_{\max} \log n) \)-wise independent binary random variables \( Y_1, \ldots, Y_{Cn^i/k_{\min}} \);

**Run** the Moser-Tardos algorithm on each table;

**Return** the current assignment;
Since there are polynomially many entries in each table, the running time of the above algorithm is \( n^{O(\Delta k_{max})} \). If both \( \Delta \) and \( k_{max} \) are constants, then the algorithm terminates in polytime. Since we proved that there is a table of size \( n \times \frac{2n \log n}{k_{min}} \) which gives a satisfying assignment, the algorithm is correct.

4 \( k \)-Universal Sets

We will briefly discuss another application of \( k \)-wise independence: \( k \)-universal sets.

A \( k \)-universal set is a set of strings, each of length \( n \), such that for any \( k \) indices \( i_1, i_2, \ldots, i_k \) and string \( s \) of length \( k \), some string \( t \) in our set has \( t[i_j] = s[j] \) for every \( 1 \leq j \leq k \). In other words, for every \( k \) indices, there are strings with every combination of bits on those indices.

Clearly there is a \( k \)-universal set of size \( 2^n \). There is also a \( k \)-universal set of size \( \binom{n}{k} 2^k \sim n^{O(k)} \).

**Application** Suppose we are launching a rocket, and each potential error depends on at most \( k \) systems in the rocket. Using a \( k \)-universal set to model each possible configuration of the systems will allow us to ensure that there is no error scenario, if we test the rocket on each string in the set.

**Fact 7.** There is a \( k \)-universal set of size \( \left(2^k \log n\right)^{O(1)} \).

**Proof.** Generate a string of length \( n \) made up of \( 2^{-2k} \)-approximate \( k \)-wise independent random bits. As we saw above, this needs \( O(k + \log \log n + 2k) \) bits of randomness. Take any \( k \) indices from this string. The probability these indices have any particular setting must be within \( 2^{-2k} \) of \( 2^{-k} \), and must therefore in particular be larger than 0.

Therefore if we iterate over all random seeds and generate our \( k \)-universal set, if we take any \( k \) indices and some setting of those bits, there will be some string with that setting on those indices.

How large is our set? We needed \( O(k + \log \log n) \) bits of randomness, so we have \( \left(2^k \log n\right)^{O(1)} \) strings in the set.

5 Error Correcting Codes for Insertions and Deletions

There are several types of noise:

1. Hamming noise, in which we have two sorts of errors:
   - (a) Symbol corruptions, which transform one symbol into another symbol, e.g. \( 1 \rightarrow 0 \) or \( B \rightarrow Z \). We count this as 2 half-errors;
   - (b) Erasures, where a symbol is replaced with a special erased symbol, often denoted \( ? \), e.g. \( 0 \rightarrow ? \). We count this as 1 half-error.

2. Synchronization errors, which have two sorts as well:
   - (a) Insertions, e.g. \( 123 \rightarrow 1253 \);
   - (b) Deletions, e.g. \( 1434 \rightarrow 434 \).

These sorts of errors may model a situation such as a CD reader, where if the motor and scanner are not perfectly synchronized, it may read one extra symbol or miss one.
Application  Hard drives are not designed for long term storage. The magnetic bits will disappear over time. But DNA can be recovered from ancient creatures, so this naturally prompts the idea of DNA storage. In this setting the DNA will be replicated by biological means which naturally insert or delete as they copy.

Recall that the rate of a code is the ratio \( \frac{u}{u+r} \) where \( u \) is the number of useful bits, and \( r \) the number of redundant bits.

Theorem 8 (Singleton bound for Hamming noise). If there are a \( \delta \)-fraction of half-errors, and a code with rate \( R < 1 - \delta - \varepsilon \), we must have an alphabet of size \( O(1) \).

Results for error-correcting codes for synchronization noise

- Schulman-Zuckerman ’99: code with \( R \) and \( \delta \) small constants.
- Guruswami et al. 2016: codes with \( R \to 1 \) as \( \delta \to 0 \), and handling \( \delta \to 1 \), if \( R = O\left( (1 - \delta)^{\frac{3}{5}} \right) \).

We will reduce the problem of synchronization noise to Hamming noise.

A first idea:

\[
\begin{align*}
X_1 & \rightarrow (X_1, 1) \\
X_2 & \rightarrow (X_2, 2) \\
\vdots & \rightarrow \vdots \\
X_n & \rightarrow (X_n, n)
\end{align*}
\]

where we number each symbol, and have our alphabet be of tuples. If there is an insertion of a symbol \((Z, k)\) at a position other than \( k \) we can ignore it, and if it was inserted in the \( k \)th position we have an erasure. Similarly deletions cause erasures, and a corruption can be caused by a deletion and insertion. Thus we have mapped one synchronization error to one half-error in the Hamming noise model.

So if we have a code for Hamming noise, we can tolerate the same fraction of synchronization errors as the code can tolerate half-errors.

But our alphabet size now increases with \( n \).

Better idea:

\[
\begin{align*}
X_1 & \rightarrow (X_1, s_1) \\
X_2 & \rightarrow (X_2, s_2) \\
\vdots & \rightarrow \vdots \\
X_n & \rightarrow (X_n, s_n)
\end{align*}
\]

where the \( s_i \), our “synchronization strings” are fixed strings over a small alphabet.

Using this method, \( k \) insertions or deletions will be mapped to \( (1 + \varepsilon)k \) half-errors, if we have synchronization strings over an alphabet of size \( \left( \frac{1}{\varepsilon} \right)^{O(1)} \).

We will pick the \( s_i \) randomly!

6 Conclusion

In this lecture we derived a deterministic version of the Moser-Tardos algorithm. We also examined \( k \)-universal sets, and began to explore error correcting codes for insertions and deletions. Next time we will finish up our discussion of error correcting codes, and prove that picking the synchronization strings randomly will provide a good code.

Acknowledgments. These notes are partially based on a previous writeup by Dabeen Lee.
References


