# 5
# The Lovász Local Lemma

## 5.1 Introduction: Beyond the Union Bound

Very often, when we want to show no "bad" event happens, we bound the probability of each bad event, and then use the trivial union: if $B_1, B_2, \ldots, B_m$ are the bad events, then the union bound states:

$$\Pr(\text{some bad event happens}) = \Pr(\cup_{i=1}^{m} B_i) \leq \sum_{i=1}^{m} \Pr(B_i). \tag{5.1}$$

Hence, if $\sum \Pr(B_i) < 1$, then $\Pr(\text{nothing bad happens}) > 0$.

### 5.1.1 Example: k-Satisfiability

Consider a $k$-CNF formula $\varphi$ with $n$ variables and $m$ clauses, where each clause has exactly $k$ literals—a literal is either a variable or its negation. For instance, here is a 3-CNF formula:

$$\varphi = (x_1 \vee \overline{x_5} \vee x_9) \wedge (x_2 \vee x_3 \vee \overline{x_{11}}) \wedge \ldots$$

We want to give conditions for when $\varphi$ *must* have a satisfying assignment.

Let's try taking a random assignment, where each variable $x_i$ is set to `True` or `False` independently and uniformly at random (u.a.r.). Let $B_i$ be the event that clause $i$ is unsatisfied by this random assignment. For a $k$-CNF clause to be unsatisfied, all $k$ of its literals must be set to the "wrong" values. Since each variable assignment is independent, the probability of this event is:

$$\Pr(B_i) = 2^{-k}.$$

Using the union bound, the probability that the formula is not satisfied by the random assignment is:

$$\Pr(\cup_{i=1}^{m} B_i) \leq \sum_{i=1}^{m} \Pr(B_i) = m \cdot 2^{-k}.$$

If $m \cdot 2^{-k} < 1$ (i.e., if $m < 2^k$), then the probability of failure is $\Pr(\cup B_i) < 1$. This implies that the probability of the complementary event—that all clauses are indeed satisfied—is greater than 0.

$$\Pr(\varphi \text{ is satisfied by random assignment}) > 0$$

By the probabilistic method, this proves that a satisfying assignment exists if $m < 2^k$.

This bound is nearly tight: indeed, if $m = 2^k$, it's possible to construct an unsatisfiable formula by taking all $2^k$ possible clauses over a set of $k$ variables: one of these clauses must be false in any assignment. Hence, a $k$-CNF formula is guaranteed to have a satisfying assignment, if and only if $m < 2^k$.

## 5.2 *Locality Constraints*

What if we have more information? Suppose we have a $k$-CNF formula where each variable appears in a "small" number, say $L$, of clauses. Given this "locality" constraint, can we satisfy formulas with more clauses than $2^k$? Surprisingly, this locality constraint gives us a lot of power:

**Theorem 5.1.** *Suppose we have a k-CNF formula where each variable appears in at most $L := 2^{k-2}/k$ clauses. Then the formula is satisfiable, independent of the total number of clauses m or variables n.*

This result follows from an important tool: the Lovász Local Lemma, which is the main topic of this lecture.

## 5.3 *The Lovász Local Lemma (LLL)*

In order to capture the notion of locality in general, we need the notion of a dependency graph. Consider a collection of "good" events $G_1, G_2, \ldots, G_m$. We define a **dependency graph** $H = ([m], E)$ on the indices of these events. The graph must satisfy the property that for any index $i \in [m]$, the event $G_i$ is mutually independent of all events $\{G_j\}_{j \notin N[i]}$, where $N[i] = \{i\} \cup \{j \mid (i,j) \in E\}$ is the *closed neighborhood* of $i$. Formally, for any $i \in [m]$ and any disjoint subsets $S, T \subseteq [m] \setminus N[i]$:

$$\Pr(G_i \mid (\cap_{j \in S} G_j) \cap (\cap_{j \in T} \overline{G_j})) = \Pr(G_i).$$

In other words, the probability of $G_i$ does not change, even if we conditioning on any collection events not in its closed neighborhood.

**Example for k-SAT**: Let $B_i$ be the bad event that clause $i$ is unsatisfied, and let the good event $G_i := \overline{B_i}$ be the complement event,

i.e., that clause $i$ is satisfied. We can define the dependency graph $H = ([m], E)$ where an edge $(i, j) \in E$ exists if clause $i$ and clause $j$ share one or more variables. The event $G_i$ depends only on the variables in clause $i$. If a set of clauses $\{G_j\}$ do not share any variables with clause $i$, then the outcome of $G_i$ is independent of the outcomes of $\{G_j\}$. This structure correctly defines a dependency graph.

### 5.3.1 *Symmetric LLL*

The Local Lemma comes from work of Erdős and Lovász (1975), and gives conditions on a collection of dependent bad events under which no bad events happen:

**Theorem 5.2** (Lovász Local Lemma, Symmetric Version)**.** *Let $G_1, \ldots, G_m$ be events with a dependency graph $H = ([m], E)$. Suppose: (i) For all $i$, $\Pr(\overline{G_i}) \leq p$, (ii) The maximum degree of $H$ is $\Delta(H) \leq d$, and that (iii) $ep(d+1) \leq 1$. Then*

$$\Pr(\cap_{i=1}^{m} G_i) > 0.$$

Observe that the "all-good" outcome holds as long as $pd \leq 1/4$, instead of the union bound condition $mp < 1$, which is much better when $m$ is large but $d$ is small. A simpler (and slightly weaker) condition than (iii) that we often use is that

$$pd \leq 1/4. \tag{5.2}$$

### 5.3.2 *Application to k-SAT*

Let's prove Theorem 5.1 using the LLL:

1. Let $G_i$ be the event that clause $i$ is satisfied by a uniformly random assignment from $\{T, F\}^n$.
2. The probability of the bad event $\overline{G_i}$ is $p := \Pr(\overline{G_i}) = 2^{-k}$.
3. The dependency graph $H$ has an edge $(i, j)$ if clauses $i$ and $j$ share variables.
4. A single clause $C_i$ has $k$ variables. Each of these variables appears in at most $L = 2^{k-2}/k$ clauses (by the theorem's assumption).
5. The degree of a vertex $i$ is the number of other clauses that share variables with clause $i$. Hence, the number of such clauses is at most $k \cdot (L-1)$.
6. So, the max degree of the dependency graph is $d \leq k(L-1) < kL = k \cdot (2^{k-2}/k) = 2^{k-2}$.
7. Let's check the LLL condition $4pd \leq 1$:

$$4pd \leq 4 \cdot (2^{-k}) \cdot (2^{k-2}) = 4 \cdot 2^{-2} = 4 \cdot \frac{1}{4} = 1.$$

Hence, by the Local Lemma, we have

$$\Pr(\cap_{i=1}^{m} G_i) = \Pr(\text{all clauses satisfied}) > 0.$$

Therefore, a satisfying assignment must exist.

## 5.4 The Algorithmic LLL

The original proof of the LLL was existential and did not provide a way to find the desired outcome. Work of Jozef Beck, and of Noga Alon, gave approaches to make some of the application algorithmic; however, these were application-dependent and somewhat cumbersome to use. A very elegant approach (and an alternate proof of the Local Lemma) was given relatively recently by later proof by Moser and Tardos (2010); this proof was algorithmic/constructive by design.

### 5.4.1 A Trivial Algorithm for k-SAT

Consider the following "trivial" algorithm for finding a satisfying assignment.

**Algorithm 1:** A Trivial Algorithm for $k$-Satisfiability

```
1.1 Start with a uniformly random truth assignment X
1.2 while there is an unsatisfied clause C do
1.3     Pick any unsatisfied clause C
1.4     Re-randomize the truth values of all variables in C
1.5 return the current truth assignment
```

This is perhaps the simplest algorithm one can imagine, and it is remarkable that such algorithm would even terminate. (Of course, it cannot terminate if the formula has no satisfying assignment.) In fact, we can show that it does so after re-randomizing only $O(m)$ times, as long as no variable appears in many clauses!

**Theorem 5.3** (Moser & Tardos)**.** *If each variable appears in less than $\frac{2^k}{ek}$ clauses, the trivial algorithm terminates in expected $O(m)$ time.*

The remaining part of this section is devoted to the proof of this theorem. The proof analyzes the expected number of resampling steps by examining the execution log of the algorithm. Let $C_1, C_2, \ldots, C_t, \ldots$ be the sequence of clauses resampled by the algorithm.

*Witness Trees* The core concept is a "witness tree," which captures the causal relationship between resampling events. For any step $t$, we construct a witness tree $T_t$ based on the history $C_1, \ldots, C_t$. The construction proceeds backwards in time:

1. Initialize $T_t$ with the root node labeled $C_t$.

2. For $i = t-1, t-2, \ldots, 1$:

   - If clause $C_i$ shares variables with any clause currently in $T_t$:
   - Add a node labeled $C_i$ to $T_t$ as a child of the **deepest** node in $T_t$ with which it shares variables. (Depth is measured from the root $C_t$. The deepest node corresponds to the most recent event in the history among those it overlaps with).
   - Otherwise, $C_i$ is discarded from this specific tree.

A witness tree is a rooted, labeled tree where if $C_i$ is a child of $C_j$, then $i < j$ and $C_i$ intersects $C_j$.

The analysis now relies on two key claims: bounding the probability that a specific tree occurs, and counting how many distinct trees are possible.

*Probability of a Specific Tree*

*Claim* 5.4. The probability that a specific witness tree $T$ of size $s$ appears in the execution log is exactly $p^s = (2^{-k})^s$.

*Proof of Claim 5.4.* For a witness tree $T$ to occur, every clause in the tree must have been unsatisfied at the moment the algorithm chose to resample it. We argue that these $s$ events are mutually independent due to the structure of the witness tree and the nature of the algorithm.

We can use a "peeling argument," analyzing the tree starting from the leaves.

Consider a leaf node $L$ in $T$. When $L$ was resampled, it must have been unsatisfied. We look at the assignment of its variables $V(L)$ at that time. If these variables had been modified by a prior resampling event $C'$ that is relevant to this causal chain, $C'$ would need to be a descendant of $L$ in $T$ (by the construction rules). Since $L$ is a leaf, it has no descendants in $T$. Therefore, the variables $V(L)$ held values that were uniformly random and independent of the other events captured in the tree (e.g., the initial assignment). The probability that $L$ was unsatisfied is exactly $p = 2^{-k}$.

Now consider an internal node $C$. By the construction of the witness tree, the children of $C$ (and all nodes in their subtrees, i.e., the clauses "below" $C$ in the tree) represent clauses that were resampled ***prior*** to $C$. While these prior events influenced the state of the variables leading up to $C$'s resampling, the algorithm ensures that when $C$ itself is processed, all its variables $V(C)$ are assigned fresh, independent, uniform random values. This is because the last time we resampled any of its variable we did so by flipping an independent coin. The probability that $C$ is unsatisfied is thus exactly $p = 2^{-k}$.

Since this independence holds for all $s$ nodes in the tree, the probability that this specific sequence of events occurs is the product of the individual probabilities:

$$\Pr(T \text{ occurs}) = p^s.$$

□

*Counting the Number of Trees* We now bound the number of distinct witness trees of size $s$.

*Claim* 5.5. The number of possible witness trees of size $s$ is at most $m \cdot (e(d+1))^s$.

*Proof of Claim 5.5.* We use an encoding scheme to count the possible trees. Assume a canonical ordering of the $m$ clauses.

1. **The Root:** There are $m$ choices for the clause at the root of the tree.

2. **Encoding the Structure and Labels:** We need to encode the remaining $s-1$ nodes and the tree structure. We utilize the constraint that a child must be a neighbor of its parent in the dependency graph. A clause has at most $d$ neighbors.

We can specify the tree structure using a preorder traversal. For any node (clause), we identify which of its neighbors (in the dependency graph) are its children in the witness tree.

We can encode the children of a node using a bit vector. Since a clause has at most $d+1$ elements in its closed neighborhood (itself and its $d$ neighbors), we can use a bit vector of length $d+1$ for each node to indicate which of these neighbors (according to the canonical order) are children in the witness tree.

We traverse the tree (e.g., in preorder) and concatenate these bit vectors. Since there are $s$ nodes in the tree, the total length of the concatenated bit string is $s(d+1)$.

The total number of edges in the tree is $s-1$. Each edge corresponds exactly to a '1' in the encoding (indicating a parent-child relationship). Thus, the encoding is a bit string of length $s(d+1)$ containing exactly $s-1$ ones.

The number of such encodings bounds the number of possible labeled tree structures (given the root):

$$\text{Number of structures} \leq \binom{s(d+1)}{s-1}.$$

We use the standard combinatorial approximation $\binom{N}{k} \leq (\frac{eN}{k})^k$:

$$\binom{s(d+1)}{s-1} \leq \left(\frac{e \cdot s(d+1)}{s-1}\right)^{s-1} = \left(e(d+1) \cdot \frac{s}{s-1}\right)^{s-1}.$$

Since $(\frac{s}{s-1})^{s-1}$ converges to $e$ from below as $s$ increases, this quantity is bounded by a constant multiple of $(e(d+1))^{s-1}$. For simplicity in the LLL analysis, we use the upper bound $(e(d+1))^s$.

Thus, the total number of possible witness trees of size $s$ is bounded by $m \cdot (e(d+1))^s$. □

*Expected Running Time* The total expected number of resampling steps is equal to the expected total number of witness trees generated during the execution (as each step $t$ generates exactly one witness tree $T_t$). We calculate this by summing over all possible sizes $s$:

$$E[\text{Total Steps}] = \sum_{s=1}^{\infty} E[\text{Number of witness trees of size } s]$$

By linearity of expectation, this is bounded by the sum over all possible trees $T$ of the probability that $T$ occurs.

$$\begin{aligned} E[\text{Total Steps}] &\leq \sum_{s=1}^{\infty} (\#\text{ of possible trees of size } s) \times (\text{Prob. of a specific tree of size } s) \\ &\leq \sum_{s=1}^{\infty} \left( m \cdot (e(d+1))^{s-1} \right) \cdot p^s \\ &= m \cdot p \sum_{s=1}^{\infty} (e(d+1))^{s-1} \cdot p^{s-1} \\ &= m \cdot p \sum_{j=0}^{\infty} (e \cdot p \cdot (d+1))^j. \end{aligned}$$

Let $r = e \cdot p \cdot (d+1)$. This is a geometric series. By the assumption of the theorem (the LLL condition), we have $r < 1$. Therefore, the series converges:

$$E[\text{Total Steps}] \leq m \cdot p \cdot \frac{1}{1-r}.$$

Since $p$ and $r$ are constants depending only on $k$ and $d$, the expected number of steps is $O(m)$.

This kind of witness tree argument is a powerful technique that has also found applications in other areas, such as analyzing the 2-choice process in load balancing.

## 5.5 *Application #2: The Beck-Fiala Problem (Discrepancy)*

Suppose we have a universe of elements $[n]$ and a collection of subsets $S_1, \ldots, S_m \subseteq [n]$. We want to partition the elements into two groups, traditionally called Red and Blue, such that every subset $S_i$ is split as evenly as possible.

We formalize this by defining a coloring $\chi : [n] \to \{-1, +1\}$, where $+1$ might represent Red and $-1$ Blue.

**Definition 5.6** (Discrepancy)**.** The **discrepancy** of a coloring $\chi$ with respect to a set system $\{S_i\}$ is:

$$\text{disc}(\chi) = \max_{i \in [m]} \left| \sum_{j \in S_i} \chi(j) \right|$$

This measures the maximum imbalance between Red and Blue elements in any set. The goal is to find a coloring $\chi$ that minimizes $\text{disc}(\chi)$.

*Motivation.* Why do we care about discrepancy? Suppose the elements $[n]$ are people, and the subsets $S_i$ represent different characteristics (e.g., gender, age brackets, education levels). If we want to divide the people into two groups (a partition), we want the groups to be balanced with respect to all these characteristics simultaneously. Minimizing the discrepancy ensures that the partition is fair across the board. This concept is fundamental in areas like experimental design and creating representative samples.

### *5.5.1 Background and the Beck-Fiala Conjecture*

In the general case, a simple randomized coloring (assigning $\pm 1$ independently and uniformly) already provides a baseline.

**Theorem 5.7.** *A random coloring $\chi$ achieves $\text{disc}(\chi) = O(\sqrt{n \log m})$ with high probability.*

(This is typically proven using the Chernoff bound and a union bound over the $m$ sets.)

However, we are often interested in cases where the set system has specific structural properties, particularly sparsity. Suppose we know that every element $j \in [n]$ belongs to at most $k$ subsets (the maximum degree of the set system).

One might hope the discrepancy depends on $k$ rather than the total number of elements $n$ or sets $m$. This intuition leads to a famous conjecture.

**Conjecture 5.8** (Beck-Fiala, 1981)**.** *If every element belongs to at most $k$ subsets, then there exists a coloring $\chi$ such that $\text{disc}(\chi) = O(\sqrt{k})$.*

This conjecture remains one of the major open problems in discrepancy theory. Beck and Fiala proved that the discrepancy is always at most $2k - 1$. There have been recent super interesting progress, by Bansal & Jiang in FOCS 2025 that resolve this conjecture when $k \geq \log^2(n)$.

### 5.5.2 *Discrepancy Bound using LLL*

The Lovàsz Local Lemma is a powerful tool for proving the existence of low-discrepancy colorings, especially when the dependencies between the sets are limited. We can use the LLL to prove a good bound under slightly stronger assumptions than required for the Beck-Fiala conjecture.

Suppose we have the following "locality" conditions:

(a) (Sparsity) Each element $j \in [n]$ belongs to at most $k$ subsets.

(b) (Uniformity) Each subset $S_i$ contains at most $k$ elements, i.e., $|S_i| \leq k$.

**Theorem 5.9.** *Under conditions (a) and (b), there exists a coloring $\chi$ such that $disc(\chi) \leq O(\sqrt{k \log k})$.*

*Proof.* We use the LLL to prove this existence. We aim for a bound $\lambda = C\sqrt{k \log k}$ for a sufficiently large constant $C$.

**1. Random Coloring:** We assign a color $\chi(j) \in \{-1, +1\}$ to each element $j$ uniformly and independently at random. These are Rademacher random variables.

**2. Bad Events and Probabilities:** Let $B_i$ be the "bad" event that subset $S_i$ has a discrepancy larger than $\lambda$. We use the Chernoff bound for the sum of Rademacher random variables:

$$\Pr(B_i) = \Pr\left(\left|\sum_{j \in S_i} \chi(j)\right| \geq \lambda\right) \leq 2e^{-\lambda^2/(2|S_i|)}$$

Since $|S_i| \leq k$ (Condition b), and substituting $\lambda = C\sqrt{k \log k}$:

$$\begin{aligned}\Pr(B_i) &\leq 2\exp\left(-\frac{C^2 k \log k}{2k}\right) \\ &= 2\exp\left(-\frac{C^2}{2}\log k\right) = 2 \cdot k^{-C^2/2}\end{aligned}$$

Let $p = 2 \cdot k^{-C^2/2}$.

**3. Dependency Graph:** We define a dependency graph $G$. An edge $(i, j)$ exists if the events $B_i$ and $B_j$ are dependent. Since the underlying variables $\chi(l)$ are independent, $B_i$ and $B_j$ are dependent if and only if they share elements, i.e., $S_i \cap S_j \neq \emptyset$.

We need to bound the maximum degree $d$. Consider a set $S_i$. We want to count how many other sets $S_j$ intersect $S_i$.

- $S_i$ has at most $k$ elements (Condition b).
- Each element in $S_i$ belongs to at most $k - 1$ ***other*** sets (Condition a).

Therefore, the maximum degree is:

$$d \leq |S_i| \cdot (k-1) \leq k(k-1) < k^2$$

**4. Applying the LLL:** We check the symmetric LLL condition: $4pd \leq 1$.

$$4pd < 4 \cdot \left(2 \cdot k^{-C^2/2}\right) \cdot k^2 = 8 \cdot k^{2-C^2/2}$$

We need to choose $C$ such that $8 \cdot k^{2-C^2/2} \leq 1$. Let's choose $C = 4$. Then $C^2/2 = 16/2 = 8$. The condition becomes:

$$8 \cdot k^{2-8} = \frac{8}{k^6}$$

This is $\leq 1$ for all $k \geq 2$ (since $8/2^6 = 8/64 \leq 1$).

Since the LLL condition is satisfied, it implies that $\Pr(\cap_{i=1}^{m} \overline{B_i}) > 0$. Therefore, there exists a coloring such that no bad event occurs, meaning every set $S_i$ has a discrepancy at most $\lambda = 4\sqrt{k \log k}$. □