the Probabilistic Method and Algorithms to find something guaranteed to exist

Example Max Cut in a graph = how close to bipartite

\[ S \subseteq V \quad \text{Partition } V \text{ into } S, V \setminus S \]

\[ \text{cut} \] to maximize \# edges between \( S \) and \( V \setminus S \)

\[ E(c(S)) \leq c(S) \]

Fact: Max Cut is NP-hard (contrast with Min Cut – poly. time)

Lemma Every graph has a cut with \( c(S) \geq \frac{1}{2} m \)

proved via randomized algorithm:

- for each \( v \), put \( v \) in \( S \) with prob. \( \frac{1}{2} \)

[Claim] \( E[c(S)] = \frac{1}{2} m \)

\[ c(S) = \sum_{u \in S} c(u,v) \]

\[ c(S) = \sum_{u \in S} \left( \sum_{v \in V} c(u,v) \right) = \sum_{u \in S} \sum_{v \in V} c(u,v) = \sum_{u \in S} \frac{1}{2} m = \frac{1}{2} m \]

Probabilistic Method: If the expected value of a random variable is \( \chi \) then there exists at least one value for the variable that is \( \geq \chi \).

For Max Cut, we conclude that every graph has a cut with \( c(S) \geq \frac{1}{2} m \)

Note: can analyze prob. above alg. gives cut \( \geq \frac{1}{2} m \)

But in fact, there is a deterministic alg. to find a cut \( \geq \frac{1}{2} m \)

Local improvement:

- Start with arbitrary \( S, V \setminus S \)

- repeat: if \( \exists v \) s.t. switching from \( S \) to \( V \setminus S \) or vice versa improves \( c(S) \) then switch.

EX: Prove final cut is \( \geq \frac{1}{2} m \)
Max SAT — Given a set of $m$ clauses (CNF)
in $n$ Boolean variables $x_1, \ldots, x_n$, find True/False assignment
to variables to make a max. number of clauses true.
e.g. $(\overline{a_1} \lor a_2) (a_1 \lor \overline{a_3}) (\overline{a}_1 \lor a_3) (\overline{a}_3)$
to make all $T$, $a_3 = F$, $a_2 = T$ but can’t get 2 middle clauses.
Max is 3.

[decision version] NP-complete — usual SAT:
NP-complete even for Max 2-SAT — clauses of size $\leq 2$
though deciding 2-SAT (can all clauses be satisfied )
is poly-time

to prove NP-complete: reduce 3-SAT to Max 2-SAT

Turn $(x \lor y \lor z)$ into $\sim 10$ 2-SAT clauses
s.t. $(x \lor y \lor z)$ is satisfied iff $\sim 7$ of the 2-SAT clauses
are satisfied.

Lemma Can always satisfy at least $m/2$ clauses
proven via:

Randomized Algorithm for Max-SAT.

Pick Truth Value Assignment at random ($\$\$)

Analysis For any clause $C$

$Pr[ C \text{ is satisfied}] = 1 - Pr[ C \text{ not satisfied}]$

$= 1 - \frac{1}{2^t}$

$t = \# \text{variables in } C$

$\geq (1 - \frac{1}{2}) = \frac{1}{2}$

$E[ \# \text{ clauses satisfied}] = \sum_{\text{clauses } C} E[C \text{ is satisfied}] \geq \frac{1}{2} m$

Can de-randomize. — may cover this later on.
Under some conditions we can guarantee that all clauses can be satisfied.

**Powerful result** - **Lovasz Local Lemma**  LLL '70's

Given events $E_1 \ldots E_m$ in a Prob. space, we want

$$\Pr [E_1 \land E_2 \ldots \land E_m] = \Pr [\neg E_i]$$

e.g. for SAT, $E_i$ = event that clause $C_i$ is satisfied

If events $E_i$ are mutually independent then

$$\Pr [\neg E_i] = \prod \Pr [E_i] > 0$$

but this totally fails in general, e.g. $C_1 = x$, $C_2 = \overline{x}$ cannot satisfy both

LLL allows limited dependence and still guarantees

$$\Pr [\neg E_i] > 0$$

We will just look at LLL for $k$-SAT -

Clauses $C_1 \ldots C_m$ where every clause has $k$ literals

For more general versions, see Motwani & Raghavan text.

For random truth-value assignment

$$\Pr [C_i \text{ not satisfied}] = \frac{1}{2^k}$$

$$\Pr [C_i \text{ satisfied}] = 1 - 2^{-k}$$

Make a dependency graph $G$ — vertices $1 \ldots m$

edge $(i,j)$ if $C_i$ and $C_j$ share a variable

e.g. $Z_1 \lor Z_2 \quad Z_1 \lor Z_3$

$Z_3 \lor Z_4$

Let $d = 1 + \max \text{ degree of } G$

**Theorem (LLL for $k$-SAT)** If $d \leq 2^{k-3}$ then all clauses can be satisfied.
Note: if each variable is in \( \leq \frac{2^{k-3}}{k} \) clauses then \( d \leq 2^{k-3} \).

Not useful for small \( k \), but e.g. \( k = 8 \Rightarrow 2^3 \), each var. in \( \leq 4 \) clauses.

The general LLL has conditions relating \( \Pr[E_i] \), \( d \leq \frac{1}{8\Pr[E_i]} \).

For decades, LLL was just an existence result.

Breakthrough 2008 Robin Moser — algorithm.

We will cover the alg. for \( k\text{-SAT} \). It is a “stupid” alg.

Alg
- pick random truth value assignment \( \Pr[x = \text{True}] = \frac{1}{2} \)
- while some clause \( C \) is not satisfied
  \( \text{FIX}(C) \)

\( \text{FIX}(C) \)
- randomly reassign the \( k \) variables in \( C \)
- while some clause \( D \) sharing variables with \( C \) is not satisfied
  \( \text{FIX}(D) \)

Note: may have \( D = C \)

What is stupid about this? It may loop forever.

We will analyze the execution tree.

Observation: If \( \text{FIX}(C) \) terminates then it finds a truth-value assignment that satisfies \( C \) and all clauses sharing variables.
Lemma. If \( \text{Fix}(c) \) terminates, it does not change any clause from satisfied to unsatisfied.

Proof. Consider clause \( c^* \). If it shares vars. with \( c \) then OK by above. Else, reassign to vars. of \( c \) doesn’t change \( c^* \) and by induction calls to \( \text{Fix}(D) \) leave \( c^* \) satisfied.

Each increases # satisfied clauses.

The top level calls make progress — there are at most \( m \) such calls but at lower levels, the number of satisfied clauses may go up and down.

Theorem. If \( d \leq 2^{k-3} \) then the Alg. terminates in poly time with high probability.

Proof based on: if \( f: A \to B \) is invertible then \( |B| \geq |A| \).

Plan: Consider truncating Alg. after \( T \) calls to \( \text{Fix} \).

Suppose \( \Pr[\text{Alg. terminates within } T \text{ calls}] = 0 \).

We will show \( T \) must be small. So for larger \( T \), \( \Pr[\] > 0.

With \( T \) calls to \( \text{Fix} \), Alg. uses \( n + k \cdot T \) random bits, so \( 2^{n+kT} \) possibilities. This is domain \( A \).

\( f: 2^{n+kT} \to \) strings encoding the alg. execution.

Encoding captures the tree: 0 to go down, 1 to go up, \( \log m \) bits to encode which clause called at top level, \( \log d \) bits to encode which clause \( D \) called in \( \text{Fix}(C) \).

- Note: at most \( d \) choices.

\( n \) bits for truth-value assignment at end.
length of encoding string:
\[ \leq n + m (\log m + 2) + T (\log d + 2) \quad \log d \leq k - 3 \]
\[ \leq n + m (\log m + 2) + T (k-1) = b \]
So \( f : \mathbb{2}^{n+kT} \rightarrow \mathbb{2}^b \)

**Claim:** \( f \) is invertible — i.e., from encoding, can recover all random bits.

**Pf of claim** From encoding we can recover sequence of clauses that \( \text{FIX} \) is called on.

Suppose last call is \( \text{FIX}(C) \).

Before this call, \( C \) was not satisfied — there’s a unique truth-value assignment for that.

We know final truth-value assignment so we know values of \( k \) variables of \( C \) before and after call.

Continuing, we can find all random values used during alg and the initial random values.

Then by \( \circ \n + kT = n + m (\log m + 2) + T (k-1) \)
\[ T \leq m (\log m + 2) \]

**Conclusion:** if \( T > m (\log m + 2) \)

then \( \Pr \left[ \text{Alg terminates} \right] > 0 \)

Can strengthen: (won’t do details, but easy enough)

If \( T > m (\log m + 2) + c \) \quad \text{poly. time.}

\[ \Pr \left[ \text{Alg terminates} \right] > 1 - \frac{1}{2^c} \]