1 Overview

In this lecture, we will learn constructive Lovasz Local Lemma.

2 Introduction

Suppose \( \varepsilon_1, \varepsilon_2, \ldots, \varepsilon_n \) are bad events and \( \Pr[\varepsilon_i] = p_i \). The goal is to make sure none of the bad events happen. There are three cases:

1. all event \( \varepsilon_i \) are independent

\[
\Pr[\bigcap_{i=1}^{n} \bar{\varepsilon_i}] = \prod_{i=1}^{n} (1 - p_i) > 0
\]

2. events \( \varepsilon_i \) can be arbitrarily correlated. In this case, by applying union bond we can get:

\[
\Pr[\bigcap_{i=1}^{n} \bar{\varepsilon_i}] \geq 1 - \sum_{i=1}^{n} p_i
\]

3. limited correlation: consider a dependency graph. connect \( \varepsilon_i, \varepsilon_j \) if they are not independent.

3 Constructive Lovasz Local Lemma(LLL)

Lovasz Local Lemma: suppose we have \( x_i \in [0, 1] \), for \( \forall i, \Pr[\varepsilon_i] \leq x_i \prod_{j: (i,j) \in E} (1 - x_j) \) then \( \Pr[\bigcap_{i=1}^{n} \bar{\varepsilon_i}] \geq \prod_{i=1}^{n} (1 - x_i) \)

4 Application

Consider a k-SAT problem

\[
C_1 \cap C_2 \cap \ldots \cap C_n
\]

where \( C_i = L_{i,1} \cup L_{i,2} \cup \ldots \cup L_{i,k} \)

Our goal is to find a satisfying assignment for the k-SAT formula. The idea is applying Lovasz Local Lemma. In this case, bad events \( \varepsilon_i \) s are equivalent to \( C_i \)s where \( C_i \) is not satisfied. Constructing the graph: connect \( \varepsilon_i, \varepsilon_j \) if they share a variable.

In LLL, if we sample \( x_i \)'s uniformly at random and \( \Pr[\varepsilon_i] = 2^{-k} \). If \( d + 1 \leq \frac{x}{\varepsilon} \) which is always satisfiable,
we can choose \( x_i = \frac{e}{2k} \), then \( x_i \prod_{j:(i,j) \in E} (1-x_j) \geq \frac{e}{2k}(1 - \frac{e}{2k})^d \geq \frac{1}{2k} \)

According to LLL, we will have:

\[
Pr[\text{satisfying assignment}] \geq (1 - \frac{e}{2k})^m
\]

4.1 Moser’s Algorithm

If \( d + 1 \leq \frac{2^k}{\text{constant}} \) we can have a efficient algorithm to solve k-SAT problem. Algorithm:

1. sample a random assignment
2. while \( \exists \) unsatisfied clause \( C_i : \text{Fix}(C_i) \)

\text{Fix}(C_i):

1. random sample variables that appear in \( C_i \)
2. while \( \exists \) unsatisfied clause \( C_j \) including \( C_i \) that is is adjacent to \( C_i : \text{Fix}(C_j) \)

Problem: The algorithm may never terminate

Claim 1: if algorithm terminates, it will find a satisfied assignment

Claim 2: if \( \text{Fix}(C_i) \) (from main loop terminates, number of unsatisfied clauses will decrease.

Proof of termination:

"Compression"

Plan: Given a random string of length

\[
total \text{ random bits} = n + sk
\]

where \( n \) is the initial solution and \( s \) is the number of calls to fix.

If the algorithm does not terminate in \( s \) calls to fix, then we can compress this string to

\[
n + m \log_2 m + S(\log_2 (d + 1) + \text{const})
\]

bits where \( S(\log_2 (d + 1) + \text{const}) \leq k - 1 \)

When \( S > m \log_2 m, n + Sk > n + m \log_2 m + S(\log_2 (d + 1) + \text{const}) \) Observation: if we know all random bits + order of Fix calls, then we will know the state of the algorithm after every Fix calls

Claim: the inverse is also true. If we know the final state of the algorithm and the order of Fix calls, we will know all the random bits which is equal to running the algorithm backwards.

In summary:

Compression: If we have the random bits we can get the sequence of Fix calls and the last state of the algorithm after \( S \) calls.

Decompression: If we have the sequence of fix calls and and the last state then we can recover the random bits.

There are two type of Fix:
1. Fix from main $\leq m$ times, $\log_2 m\text{bits}$

2. Fix from Fix $\log_2 (d + 1) + \text{Const}$ bits per call