The Probabilistic Method and Entropy Compression

Ziyang Jin[1]

[1]Theory Group
Department of Computer Science
University of Toronto

26 April 2023

Note

This talk is partially based on:

- (Book) *The Probabilistic Method* by Alon and Spencer, 1992
- (Book) *Graph Colouring and the Probabilistic Method* by Molloy and Reed, 2001
- (Blog) *Moser's entropy compression argument* by Terence Tao, 2009
- (Talk) Tutte's 100th Distinguished Lecture Series by Molloy, 2017

Table of Contents

The Probabilistic Method

[Wikipedia] The probabilistic method is a nonconstructive method, primarily used in combinatorics and pioneered by Paul Erdős. In 1947, Erdős used it to give a lower bound on the Ramsey number.

The Probabilistic Method

[Alon, Spencer 1992] The basic probabilistic method can be described as follows: in order to prove the existence of a combinatorial structure with certain properties, we construct an appropriate probability space and show that a randomly chosen element in this space has the desired properties with positive probability.

Ramsey Number

$R(k,k)$ the minimum integer n such that every red-blue edge coloring of K_n contains either a blue clique or a red clique of size k.

Theorem

For any integer n

$$R(k,k) > n - \binom{n}{k} 2^{1-\binom{k}{2}}$$

Proof. Random 2-color edges of K_n. For any set R of k vertices, let X_R be the indicator variable that the induced subgraph of K_n on R is monochromatic.

$$Pr(X_R = 1) = 2 \times (\frac{1}{2})^{\binom{k}{2}} = 2^{1-\binom{k}{2}}$$

UNIVERSITY OF TORONTO

Ramsey Number

Proof. (continued) Let $X = \sum X_R$. Then by linearity of expectation

$$E[X] = \binom{n}{k} 2^{1-\binom{k}{2}}$$

Thus there exists a 2-coloring for which $X \leq \binom{n}{k} 2^{1-\binom{k}{2}}$. Fix such a coloring. For each monochromatic k-set, remove one vertex. So at most $\binom{n}{k} 2^{1-\binom{k}{2}}$ will be removed (because the same vertex may be removed multiple times). Thus, the induced subgraph formed by the remaining vertices has no monochromatic k-set. $\square$

Ramsey Number

So we have the result:

$$R(k,k) > n - \binom{n}{k} 2^{1-\binom{k}{2}}$$

If we choose $n \sim e^{-1} k 2^{k/2}(1 - o(1))$, we can get

$$R(k,k) > \frac{1}{e}(1 + o(1)) k 2^{k/2}$$

[Wikipedia] "Erdős asks us to imagine an alien force, vastly more powerful than us, landing on Earth and demanding the value of $R(5,5)$ or they will destroy our planet. In that case, he claims, we should marshal all our computers and all our mathematicians and attempt to find the value. But suppose, instead, that they ask for $R(6,6)$. In that case, he believes, we should attempt to destroy the aliens." – Joel Spencer

Lovász Local Lemma

Theorem (Erdős, Lovász 1975)

Consider a set $\mathcal{E}$ of (bad) events such that for each $A \in \mathcal{E}$
(a) $Pr(A) \leq p < 1$
(b) A is mutually independent of a set of all but at most d of the other events
If $epd \leq 1$ then with positive probability, none of the events in $\mathcal{E}$ occur.

Lovász Local Lemma

Theorem (Erdős, Lovász 1975)

Consider a set $\mathcal{E}$ of (bad) events such that for each $A \in \mathcal{E}$
(a) $Pr(A) \leq p < 1$
(b) A is mutually independent of a set of all but at most d of the other events
If $epd \leq 1$ then with positive probability, none of the events in $\mathcal{E}$ occur.

Above is the symmetric version of the local lemma, there is also Asymmetric Local Lemma, Weighted Local Lemma, General Local Lemma, Lopsided Local Lemma.

Example: k-SAT

The k-SAT formula has n variables and m clauses, and each clause has k literals. Moreover, each clause shares at least one variable with at most other d clauses.

Theorem

If $d \leq \frac{2^k}{e}$, then the k-SAT formula is satisfiable.

How do we use Lovász Local Lemma to prove this theorem?

Example: k-SAT

The k-SAT formula has n variables and m clauses, and each clause has k literals. Moreover, each clause shares at least one variable with at most other d clauses.

Theorem

If $d \leq \frac{2^k}{e}$, then the k-SAT formula is satisfiable.

How do we use Lovász Local Lemma to prove this theorem?

We can view the k-SAT formula as a hypergraph. Each clause C_i is dependent with other d clauses, and is mutually independent with other $m - d$ clauses.

Example: k-SAT

Proof.
Independently assign each variable $x_1, ..., x_n$ true or false with $\frac{1}{2}$ probability.
Bad event A_i: clause C_i is unsatisfied.

$$Pr(A_i) = p = (\frac{1}{2})^k$$

The set of events $\mathcal{E} = \{A_1, A_2, ..., A_m\}$. The Lovász Local Lemma tells us as long as

$$epd = e \cdot (\frac{1}{2})^k \cdot d \leq 1 \Rightarrow Pr(\text{no clause is unsatisfied}) > 0$$

the k-SAT formula can be satisfied. The biggest d we can take here is $\frac{2^k}{e}$. □

An Algorithm to Find the Assignment

Lovász Local Lemma gives only nonconstructive proofs. If we know something exists, how do we find an algorithm to find it?

For example, if you know a truth assignment for k-SAT exists, how do you design an algorithm to find it?

Moser's algorithm

k-SAT: Each clause intersects at most $d \leq 2^{k-c}$ other clauses, where c is a (typically small) constant. Note that previous $d \leq \frac{2^k}{e}$, we make it $d \leq 2^{k-c}$ here to make the analysis a little bit easier to understand.

FIX

```
While there are any unsatisfied clauses
    pick an arbitrary unsatisfied clause
    randomly reassign the variables in that clause independently
```

How to prove this algorithm converges?

Moser's algorithm

k-SAT: Each clause intersects at most $d \leq 2^{k-c}$ other clauses, where c is a (typically small) constant. Note that previous $d \leq \frac{2^k}{e}$, we make it $d \leq 2^{k-c}$ here to make the analysis a little bit easier to understand.

FIX

```
While there are any unsatisfied clauses
    pick an arbitrary unsatisfied clause
    randomly reassign the variables in that clause independently
```

How to prove this algorithm converges?

If the k-SAT formula is unsatisfiable, the algorithm will run forever. The formula is satisfiable due to the condition that $d \leq 2^{k-c}$. How do we utilize this condition in proving this algorithm terminates?

Where the name comes from

The name "Entropy Compression" was given by Terence Tao in his blog *Moser's entropy compression argument*.

[Tao 2009]

A basic strategy to ensure termination of an algorithm is to exploit a monotonicity property, or more precisely to show that some key quantity keeps increasing (or keeps decreasing) with each loop of the algorithm, while simultaneously staying bounded ... Recently, a new species of monotonicity argument was introduced by Moser, as the primary tool in his elegant new proof of the Lovász local lemma. This argument could be dubbed an *entropy compression* argument, and only applies to probabilistic algorithms which require...

Key Idea of Entropy Compression

[Tao 2009]

...Thus, each stage of the argument compresses the information-theoretic content of the string $A + R$ into the string $A' + R' + H'$ in a lossless fashion. However, a random variable such as $A + R$ cannot be compressed losslessly into a string of expected size smaller than the Shannon entropy of that variable...

Key Idea

You cannot (losslessly) represent a string of random bits with a shorter string

The randomized algorithm requires a random string of bits generated by a RNG as a resource. Our goal is to find a lossless compression of the random string of bits generated by the RNG.

Entropy Compression

We call it a <u>*step*</u> when RNG generates k random bits.

Benchmark: If the algorithm runs for t steps, then the random string should have tk bits.

Goal: as $t \to \infty$, find a lossless compression string $\phi(t)$ of the random string such that $tk - |\phi(t)| \to \infty$. (or as long as $\phi(t)$ is small enough to beat the Shannon entropy)

If we are able to find such a compression, then the number of steps t cannot increase forever (otherwise it would be a contradiction to the "Key Idea"), thus the algorithm must terminate.

Entropy Compression

k-SAT: Each clause intersects at most $d \leq 2^{k-c}$ other clauses.

FIX

```
While there are any unsatisfied clauses
    pick an arbitrary unsatisfied clause
    randomly reassign the variables in that clause independently
```

Random string of bits: suppose the while-loop runs for t iterations, each iteration it requires k random bits. In total, it requires tk bits.

History File

k-SAT: Each clause intersects at most $d \leq 2^{k-c}$ other clauses.

FIX (with log)

```
While there are any unsatisfied clauses
    pick an arbitrary unsatisfied clause $C_i$
    randomly reassign the variables in that clause independently
    write down $C_i$ to the log
```

After t steps, our log looks like: $C_7, C_{12}, C_3, C_{12}, \ldots, C_{41}, C_{227}$.

Let σ_t be the truth assignment of all random variables $x_1, x_2, \ldots, x_n$ after t steps, like a "snapshot". Are we able to recover the random string of bits using the log history and σ_t?

History File

After t steps, our log looks like: $C_7, C_{12}, C_3, C_{12}, ..., C_{41}, C_{227}$. Let σ_t be the truth assignment of $x_1, x_2, ..., x_n$ after t steps.

Claim

We can recover every step of the algorithm (i.e. the random bits) using the log and σ_t.

How? To obtain σ_{t-1}, we just need to take σ_t and falsify every literal in C_{227}.

e.g. In order to get σ_{t-1}, take $C_{227} = (\overline{x}_2 \vee x_7 \vee x_{12})$, then set $x_2 = T, x_7 = F, x_{12} = F$ and keep everything else in σ_t.

Entropy Compression

Random string: tk bits
Our representation: $\{C_7, C_{12}, C_3, C_{12}, ..., C_{41}, C_{227}\} + \sigma_t$

Do we get a compression?

Entropy Compression

Random string: tk bits
Our representation: $\{C_7, C_{12}, C_3, C_{12}, ..., C_{41}, C_{227}\} + \sigma_t$

Do we get a compression? Not really!

σ_t is n bits $\Rightarrow$ t needs to be big enough to make n bits negligible.

Suppose we have m clauses, then we need $\log m$ bits to represent the clause in each step.

In total, we need $t \log m + n$ bits $\Rightarrow$ not necessarily shorter than tk since it is possible that $m \geq 2^k$.

A More Careful Compression Strategy

Random string: tk bits
Our representation: $\mathsf{NumOfBits}(\{C_7, C_{12}, C_3, C_{12}, ..., C_{41}, C_{227}\}) + n$ bits

Key Observation

Small degree provides an efficient way to name the clauses.

A More Careful Compression Strategy

Random string: tk bits
Our representation: $\text{NumOfBits}(\{C_7, C_{12}, C_3, C_{12}, \ldots, C_{41}, C_{227}\}) + n$ bits

Key Observation

Small degree provides an efficient way to name the clauses.

Solution: Instead of writing down C_{227}, we can write 8 meaning "Fix the 8th neighbour of the previous clause". (Suppose C_{227} is the 8th neighbour of clause C_{41}).

For each clause, we can label the neighbours of that clause from 1 to d. Now we just need $\log d$ bits to represent each clause.

A More Careful Compression Strategy

Random string: tk bits
Our representation: $(t \log d + n)$ bits?

It seems we are able to save some bits now as $d \leq 2^{k-c}$. Are we done yet?

A More Careful Compression Strategy

Random string: tk bits
Our representation: $(t \log d + n)$ bits?

It seems we are able to save some bits now as $d \leq 2^{k-c}$. Are we done yet? Not really. We saved the bits but not able to recover the random string of bits.

- The very first clause you note down cannot be written as a neighbour of some clause, so we need to write down C_7, which takes $\log m$ bits. Also, you might need to "jump back" to one of the originally failed clauses. Suppose there are m' unsatisfied clauses in the beginning. Note that $m' \leq m$, so in total you need an extra $m' \log m$ bits.

[Log] Fix C_7; Fix 5th neighbour; Fix 8th neighbour; ....; Fix C_{88}; Fix 2nd neighbour; ...

Entropy Compression

Random string: tk bits
Our representation: $\leq (t \log d + m \log m + n)$ bits?

Are we done yet?

Entropy Compression

Random string: tk bits
Our representation: $\leq (t \log d + m \log m + n)$ bits?

Are we done yet? Not really.

- (Fixed) You might need to "jump back" to one of the originally failed clauses. Suppose there are m' unsatisfied clauses in the beginning. Note that $m' \leq m$, so in total you need an extra $m \log m$ bits.
- It does not have to jump back to the "root", it might need to fix something in the middle of the tree.

Entropy Compression

Random string: tk bits
Our representation: $\leq (t \log d + m \log m + n)$ bits?

Are we done yet? Not really.

- (Fixed) You might need to "jump back" to one of the originally failed clauses. Suppose there are m' unsatisfied clauses in the beginning. Note that $m' \leq m$, so in total you need an extra $m \log m$ bits.
- It does not have to jump back to the "root", it might need to fix something in the middle of the tree.

Solution: Write down "Return" to the log

Entropy Compression

Modified algorithm. FIX1 deals with the initially failed clauses.

FIX1:

```
while there are any unsatisfied clauses
    pick an arbitrary unsatisfied clause $C_i$
    write down $C_i$ to log
    call FIX2($C_i$)
```

FIX2 does the “pointing to neighbour” business.

FIX2(C):

```
randomly reassign the variables in clause $C$ independently
while some neighbour of clause $C$ is unsatisfied
    pick an arbitrary unsatisfied neighbour ($i$th neighbour)
    write down “Fix $i$th neighbour”
    Let $C'$ be the $i$th neighbour of $C$, call FIX2(C')
write down “Return”
```

Entropy Compression

[Log]
step 1: Fix C_7;
step 2: Fix 5th neighbour; (C_{27})
step 3: Fix 8th neighbour; (C_{69})
Return; (back to C_{27})
step 4: Fix 25th neighbour; (C_{77})
Return; (back to C_{27})
Return; (back to C_7)

Observation: There are at most t returns in t steps.

Each "Return" can be encoded by a constant number of bits.

Entropy Compression

Random string: tk bits
Our representation: $\leq t(\log d + \mathrm{O}(1)) + m \log m + n + \mathrm{O}(1)$ bits

Therefore, as long as t is large enough, $(m \log m + n + O(1))$ will be negligible.

As long as $\log d + O(1) < k$, we "save" some bits at each step. Eventually, our representation will become shorter than the random string of bits.

Are we done yet?

Entropy Compression

Random string: tk bits
Our representation: $\leq t(\log d + \mathrm{O}(1)) + m \log m + n + \mathrm{O}(1)$ bits

Therefore, as long as t is large enough, $(m \log m + n + O(1))$ will be negligible.

As long as $\log d + O(1) < k$, we "save" some bits at each step. Eventually, our representation will become shorter than the random string of bits.

Are we done yet? Let's check if we can recover the random string of bits.

Entropy Compression

Claim: Our representation is a lossless compression of the string of random bits.

Goal: Recover the string of random bits.

How?

Entropy Compression

Claim: Our representation is a lossless compression of the string of random bits.

Goal: Recover the string of random bits.

How? A two-pass algorithm:

1. (forward-pass) Go from the beginning of the log to create a sequence of clauses getting fixed at each step: $C_{i_1}, C_{i_2}, \ldots, C_{i_t}$.
2. (reverse-pass) Use σ_t and C_{i_t} to recover σ_{t-1}, and then recover σ_{t-2}, and so on...

Conclusion

We are able to find a lossless compression on the random string of bits. Thus the randomized algorithm must terminate (with high probability).

Remark

Note that every step, it only saves a very tiny amount of bits (usually constant or logarithm number of bits), but eventually the savings will cover the overhead and thus we get a compression.

Q & A

Questions?

Thank you

Thank you!