Randomized Algorithms 2020-1
Lectures 7-8

Card Guessing and the Lovasz Local Lemma

Moni Naor

1 Card Guessing

The scenario we are considering is where a deck of n distinct cards (for simplicity labeled $1, 2, \ldots, n$) is shuffled and the cards from the deck are drawn one by one. A player called 'guesser' tries to guess the next card, for n rounds and gets a point for each correct guess. We are interested in the expected number of points the guesser can have.

Suppose that the guesser has *perfect memory* and can recall all the cards that it has seen, then what is expected number of correct guesses? At any point the guesser picks one of the cards that have not appeared so far as a guess. If there are i cards left, the probability of guessing correctly is $1/i$ and the expected number of guesses is

$$1 + \frac{1}{2} + \frac{1}{3} \ldots + \frac{1}{n} = H_n \approx \ln n.$$

Note that any guess of an unseen card has the same probability of success, so there is really not much of a strategy here.

Question: What can you say about high concentration in this case?

Now consider the opposite scenario, where the guesser has no memory at all. I.e. before it turns over a card it has no idea what cards have already appeared. But we will give it for free the round number. So the best strategy it may have is represented by a fixed guess g_i for the round i. The probability that this is correct is $1/n$, so the expectation over of all n rounds is 1.

Note: By guessing, say, '1' all the time the guesser can assure getting exactly 1 correct guess.

Now suppose that the guesser has m bits of memory. That is it has a state M represented by m bits; whenever it sees a card that was turned over it moves to the next state and outputs a guess. What is the expected number of points now? How many bits do we need in order to achieve something like what the perfect-memory-guesser can achieve.

One observation is that we can pretend as if there are only m cards, i.e. ignore all cards of face value larger than m and have a perfect recall on the first m cards. This gives us $H_m \approx \ln m$. Is this the best possible? As we will see, you can do much better.

Remember the last k cards: It is possible to know for certain the last card using $\log n$ bits of memory: keep track of the sum of the cards $(\text{mod} n)$. Just before the last card arrive, one can figure out the unique missing one. We can generalize this to the k last cards using $O(k \log n)$ bits. For instance, the polynomial from the set comparison protocol. We track the value of the polynomial at k points, starting with the value of the points of the set $A = \{1, 2, \ldots n\}$.[1] Can use this to get $H_{m'}$ on expectation for $m' = m/\log n$.

The two methods are compatible: of the m bits: use $m/2$ for the first method and $m/2$ for the second one. The last card with face value in $1, ..., m/2$ is expected when there are $2n/m$ cards left. Or the expected number of cards with face value at most $m/2$ appearing in the the last $m/2 \log n$ cards is $\frac{m^2}{2} \log n < 1$. So for $m \leq \sqrt{n}$ the two useful periods do not overlap! We get $\min\{2H_{m'}, H_n\}$. For $s = \sqrt{n}$ this is almost as much as for $m = n$.

As we will see it is possible to do much better.

2 Low Memory Card Guessing by Following Subsets

Claim 1. *There is a method using m bits of memory that obtains $1/2 \min\{\log n, \sqrt{m}\}$ correct guesses in expectation*

In other words, with $\log^2 n$ bits of memory you get the close to the maximum benefit. Furthermore, you can obtain expectation $(1 - \alpha) \ln n$ with m which is $O(\log(1/\alpha) \log^2 n$.

In terms of the code used, you can get away with just the simple idea of summing the cards. The general idea is to follow the cards that appeared in various subsets of $[n]$. For each such subset use two accumulators:

1. Sum of the values of the cards seen so far.
2. Number of the cards from the set seen so far.

The memory needed for the two accumulators is just $O(\log n)$ bits. In fact, if the sets are of size w, then just $2 \log w$ bits are need, since you can do all the computation mod w.

If there is such a subset where all but one card appeared, then (a) We can detect it (b) The card is a reasonable guess, in the sense that we know that it has not appeared before.

What if there are two such subsets? Then given that we assumed that the order of the cards is random, it does not really matter which one is guessed (at least not for the expectation).

[1] In more detail, consider the polynomial $P_A(x) = \Pi_{a \in A}(x - a)$ on the set $A = \{1, 2, \ldots, n\}$ over a finite field of size at least $n + k + 1$. The algorithm is:

- Pick $k + 1$ many points x: $n + 1, n + 2, \ldots, n + k + 1$, Evaluate $P_A(x)$ at these points and store each value separately.
- As card y goes by: divide the value at point $x = n + i$ by $(n + i - y)$. This assures that the value stores for x is $P_{A'}(x)$ for A' which is the set of cards that have not appeared yet.
- When k cards are left: reconstruct the degree k polynomial using the $k + 1$ points, and recover the remaining values.

Subset construction: We consider all the subsets of the form $[1-w]$ for $w = 2^i$. I.e. the subsets are:

$$[1-1], [1-2], [1-4], [1-8], [1-16], [1-32], \ldots, [1-n]$$

If there is a subset (range) where a single card is missing, then this card is the current guess.

Property: In this construction there cannot be competing good cards to guess. For all k and k': if card j is the missing one from the set $[1-k]$, then there cannot be a different one missing from the set $[1-k']$.

Call a subset $[1, w]$ **useful** if the last card from it that appears is *not* the last card in the next subset $[1, 2w]$. Each subset contributed a good guess, but it could be that several subsets contributed the same guess. However, if a subset is useful, then it is the only one to whom we attribute the good guess. So the number of good guesses is simply the number of useful subsets. The probability that a subset $[1, w]$ is useful is precisely the probability that in the 'next' subset $[1, 2w]$ the last card does not come from $[1, w]$. This is $(2w - w)/2w = 1/2$.

The expected number of useful subset is therefore $1/2 \log n$ and this is also the expected number of good guesses.

One idea for improving this would be the to have the subsets (ranges) be denser (and have more subsets). Suppose that ratio between two successive ranges is $1 + \gamma$. Then there are $\log_{1+\gamma} n$ such subsets. The probability of a set being useful now (i.e. that its last member arriving does not belong to a subset that contains it) is $\gamma/(1 + \gamma)$. The expected number of useful sets is

$$\frac{\gamma}{1+\gamma} \log_{1+\gamma} n = \frac{\gamma}{(1+\gamma) ln(1+\gamma)} \ln n.$$

This goes to $\ln n$ as γ goes to zero.

We can also add other subsets and keep track of them. The analysis will be harder. We can also deduce missing cards by adding a few equations together.

A Different Approach: We will consider what are the chances that when we have i cards left, for $1 \leq i \leq n$ that there is a *reasonable* guess at this point - a card that we know for certain that has not appeared before. In other words, that there is a subset with a single missing card.

Suppose that we construct random subsets with various probabilities: For each $1 \leq j \leq \log n$ we construct a subset S_j by picking each element independently with probability $p_j = 2^j/n$. At step i what is the chance that there is a reasonable candidate? Let T_i be the set of unseen cards at the point where i cards are left. To yield a reasonable guess i steps from the end, subset S_j should intersect the set T_i at exactly one point. Pick $\lfloor \log n/i \rfloor \leq j \leq \lceil \log n/i \rceil$. Now

$$\Pr[|S_j \cap T_i| = 1] = i \cdot \frac{2^j}{n} \left(1 - \frac{2^j}{n}\right)^{i-1}.$$

Note that this assumes full Independence of the choice of S_j. For this value of j, we have that $i \cdot \frac{2^j}{n}$ is between $1/2$ and 1 and $(1 - \frac{2^j}{n})^{i-1}$ is roughly $1/e$. So we get what we wanted with some constant probability, We have simple amplification in this case: we can use more sets for each j. Since each set is chosen independently, the probability of failure goes down with the number of subsets. For any $\alpha > 0$ we can get to probability $1 - \alpha$ of at least one subset succeeding. The expectation of good guesses in this case would $(1 - \alpha) \ln n$ using $O(\log(1/\alpha) \log n$ subsets.

In terms of memory, we will be using altogether $m \in O(\log(1/\alpha) \log^2 n)$ bits.

We analyzed this case when the construction of S_j was made via a -wise independent hash function.

Low Memory Case: What can we do if m is small, say $m \leq \log^2 n$? In this case we can get expectation of $\sqrt{m}$ good guesses. We treat the domain as if it is of size $2^{\sqrt{m}}$, and ignore all other cards! Now $m = \log^2(2^{\sqrt{m}})$ and therefore we have enough memory to run the previous guesser.

3 Bounds on Best Possible Guessers

We show that the guessers of the previous section are the best possible low memory guessers, up to constants.

Claim 2. *Suppose that $m \leq \log^2 n$ and $\beta > 0$. Then any algorithm using m bits of memory can get on expectation (over the cards) at most*

$$\beta \ln n + (1 - \beta) \frac{m}{\ln n}$$

correct guesses. For $\beta = (\sqrt{m}/\ln n)$ this is $2\sqrt{m}$.

Proof idea: Use the guessing algorithm to encode an ordered set of size of t elements of of n with fewer than $\log(n(n-1 \cdots n-t+1))$ bits. We will save around the order of the number of correct guesses ℓ in last t steps using only m bits of memory.

Proof by Compression: a quite general method to prove success of an algorithm by showing that failure allows us to compress the random bits used. We know that for any method the probability of compressing and chopping off w bits from a random string is 2^{-w}. In such a proof we consider a random string used in the system. This could be the random function that should be inverted in the example of function inversion or the randomness used by an algorithm (this will be the case with the Local Lemma). We then show that the event we want to bound implies an encoding of the randomness which compresses the representation. We know that the probability of compressing w bits (i.e. coming up with a representation that is w bit shorter than the length in a way that allows us to retrieve the original string) is at most 2^{-w}. This allows us to bound the probability of the bad event by 2^{-w}.

Example (not directly related): the "birthday paradox". Suppose that you have k random elements $x_1, x_2, \ldots x_k$ from a domain of size n. When, can you expect collision, i.e. for what value of k as a function of n. We note that if $x_i = x_j$, then it is possible can encode x_j by pointing out to i (as is done in the Lempel-Ziv family of compression algorithms). In such a case, instead of using $\log n$

bits to encode x_j we need only $\log k + \log k$ bits. This saves (i.e. compresses) when $k < \sqrt{n}$, so we conclude that it is not likely to happen before $k \geq \sqrt{n}$.

Homework. Let $f\colon \{0,1\}^n \mapsto \{0,1\}^n$ be a random function. Prove via compression that any algorithm that has **black-box access** to f and receive $y \in \{0,1\}^n$ and tries to find $x \in \{0,1\}^n$ s.t. $f(x) = y$ must access f close to 2^n times.

The method can be used to prove the run time of the algorithm in the "Algorithmic Lovasz Local Lemma" and the success probability of Cuckoo Hashing.

Proof of claim: Set $t = n^{1-\beta}$. Simulate the guesser on a deck of card, where the first $n-t$ cards are from $[n]\setminus T$ and the t cards are ordered according to T. Record its memory (m bits) after the first $n-t$ steps and from that point on see when the guesser gives correct guesses. These can be used to help describe T. Let the number of correct guesses be L (this is a random variable)[2]. Then to record T, we note the location of the L places with a correct guess and provide the remaining $t-L$ missing values. So how many possibilities do we have? For the memory 2^m, for the good locations $\binom{t}{L}$ and for the other values an ordered set of size $t-L$ out of n.

The contradiction comes from counting the number of the ordered sets T in two different ways: $n(n-1)\cdots(n-t+1)$ all the possible options for ordered T and $\binom{t}{L}\cdot n(n-1)\cdots(n-t+L+1)2^m$ - upper bound on the possible options for ordered T according the encoding. So we have

$$
\begin{aligned}
n(n-1)\cdots(n-t+1) &\leq \binom{t}{L}\cdot n(n-1)\cdots(n-t+L+1)2^m && (1)\\
\therefore (n-t+L)(n-t+L-1)\cdots(n-t+1) &\leq \binom{t}{L}\cdot 2^m && (2)\\
\therefore (n-t)^L &\leq t^L\cdot 2^m && (3)
\end{aligned}
$$

Taking logs we get

$$L \leq \frac{m}{\ln(n-t)-\ln t} \approx \frac{1}{\beta}\cdot\frac{m}{\ln n}.$$

Suppose that the guesser is perfect in the first $n-t$ steps, in the sense that all the guesses are reasonable. Then the expected number of correct is is $H_n - H_t = \beta \ln n$. So we get that the total number of guesses is not expected to be better than $\beta \ln n + \frac{1}{\beta}\cdot\frac{m}{\ln n}$. Taking the best β to be $\sqrt{m}/\ln n$, we get that this is not better than $2\sqrt{m}$.

Note that this bound still holds even if the guesser has at it disposal a large amount of randomness that it can repeatedly access (i.e. storing the randomness is not charged to the memory). So we conclude with tight bounds up to constants:

Theorem 1. *There is a guesser using m bits of memory that obtains $1/2\min\{\ln n, \sqrt{m}\}$ correct cards in expectation and any guesser using m bits of memory can get* at most $O(\min\{\ln n, \sqrt{m}\})$ *correct cards in expectation.*

[2] Question: why is it *sort* of ok to refer to it as if it is a fixed value?

4 Mirror Games

We also discussed Mirror Games and their relationship to card guessing [3, 2]. These are two player games where in the simplest case there is an even number of cards. The players take turns saying a name of a card and a player loses if this card was mentioned already by either one of the players. If all cards are finished, then it is a draw. The second player has a low memory strategy by mirroring the first player: fix a matching on the cards (e.g. the cards with the last bit flipped). For every move by the first player, respond by the matched card. You can view the matching as providing a subset in our setting and no bits need to be spent on it, since the current card indicates that the other has not arrived yet.

What Garg and Schnieder showed is that there is no deterministic sublinear strategy for the first player that is guaranteed to draw. The question is when can the first player have a reasonable chance of achieving a draw with little memory. Since it is a game, it is adversarial in nature, but on the other hand, half of the sequence itself is determined by the other player.

We mentioned that if there is a secret matching available to the first player, then this player can use the matching most of the time, but from time to time will need to find and unused pair (in case the second player hits the singleton). Te first player can apply the techniques of card guessing in order to find out a vacant pair. The expected number of times this happens is roughly $1/2H_{2n}$.

5 The Local Lemma

The *Union Bound* allow us to argue that certain events have a non-zero probability of not occurring simultaneously. Consider events $U = \{A_1.A_2, \ldots A_m\}$. For any event A_i the dependency on the other events is arbitrary. We know that

Theorem 2. *If there are $0 \leq p_i \leq 1$ s.t. $\Pr[A_i] \leq p_i$ and $\sum_{i=1}^{m} p_i < 1$, then the probability that no event A_i happens is positive.*

If all the $p_i = p$ then we need that $mp < 1$.

An example of an application is a CNF formula with m clauses where each clause has more than $\log m$ literals. Such a formula is necessarily satisfiable. Event A_i is clause i is not satisfied. $\Pr[A_i] < 1/m$.

The Local Lemma allows us to make such arguments based on local considerations only. You can read about in in either Alon-Spencer or Mitenmacher-Upfal.

We discussed two applications of the lemma. One was to argue that a CNF formula where every clause is of size k and every variable appears at most $2^k/ke$ clauses is necessarily satisfiable.

The classical proof is non-constructive and for instance does not yield an algorithm that finds a satisfying assignment to a CNF. Next time we will see the algorithmic version of Moser and Tardos.

6 Hat Guessing

We saw an application of the lemma to hat guessing. Hat guessing puzzles are very common, and in recent years a variant related to graphs has become popular.

The following is taken almost verbatim from [4]. Suppose that n players are positioned on the vertices of a finite, simple graph G. An adversary puts a **colored** hat on each of their heads, in one of q colors. The players can only see the hats on their (immediate) neighbors' heads, and in particular no player sees their own hat. The players simultaneously guess the colors of their own hats, and they collectively win if any single one of them guesses correctly. The players may not communicate after the hat colors are assigned but may agree upon a strategy beforehand (and it could be a different strategy for every node). The hat guessing number $HG(G)$ of G is then the largest q for which the players have a winning strategy in the game with q colors.

If there is at least one edge in the graph, then $HG(G) \geq 2$. Why?

The classic case is when $G = K_n$, the complete graph on n vertices, and in this case $HG(K_n) = n$. A winning strategy is for the i-th player to guess the hat color (identifying colors with values $\bmod n$) that would make the total of all the colors sum to $i \bmod n$. *Exactly one* player is correct in its sum and hence this player will be correct in its guess. Also for trees T it is known that $HG(T) = 2$, see Butler, Hajiaghayi, Kleinberg, and Leighton[1].

We showed that in any graph with maximum degree Δ we have that $HG(G) \leq e\Delta$. In other words, for every such graph, for every coloring strategy (which may specify for each node a different strategy) show that there is way to color the nodes with no more than $e\Delta$ colors such that the strategy fails: no node guesses its color correctly.

In more detail, let $G = (V, E)$ be a graph and let the neighborhood of a node $v \in V$ be $N(v)$. The strategy of the players is specified by a collection of functions, where for every node v there is a mapping from the colors of the neighbors to a color (that is the guess given that the neighbors are colored in certain way), $f_v : [q]^{|N(v)|} \mapsto [q]$. For any possible coloring with q colors of the nodes in V the strategist wants that there will be at lest one node v colored c with neighbors colored $c_1, c_2, \ldots c_{|N(v)|}$ s.t. $f_v(c_1, c_2, \ldots c_{|N(v)|}) = c$.

We showed that if $q > e\Delta$ then there is *no* such strategy by the players.

References

[1] S. Butler, M. T. Hajiaghayi, R. D. Kleinberg, and T. Leighton, *Hat guessing games*, SIAM J. Discrete Math.22(2008), 592–605.

[2] U. Feige, A randomized strategy in the mirror game, https://arxiv.org/pdf/1901.07809.pdf

[3] Sumegha Garg and Jon Schneider, The space complexity of mirror games, ITCS 2019.

[4] Hat Guessing Numbers of Degenerate Graphs, `https://arxiv.org/abs/2003.04990`

[5] Boaz Menuhin and Moni Naor, Keep That Card in Mind: Card Guessing with Limited Memory, ITCS 2022.