Statistical Physics for Communication and Computer Science 24 Feb. 2011

Lecture Notes 1: Models and Questions

Lecturer: Lecture was given by prof. Urbanke *Scribe:Mahdi Jafari*

1 Introduction

In this lecture we will be going to discuss about the main objective of this course. We start this course by introducing two different types of problems; the satisfiability problems and the coding problems, and show that although these are two completely different problems there are exists some similarities in their nature. To this end, we use different tools from statistical physics in order to understand and analyze the aforementioned problems.

2 Satisfiability Problem

Suppose we are given a set of n Boolean variables $\{x_1, \ldots, x_n\}$. Each variable x_i can takes on the values 0 and 1, where 0 means "false" and 1 means "true". We define a *literal* to be either a variable x_i or its negation $\bar{x}_i$. A *clause* is a disjunction of literals, e.g., $C = x_1 \vee x_2 \vee \bar{x}_3$ where the operator "$\vee$" denotes the Boolean "or" operator. An *assignment* is an assignment of values to the Boolean variables, e.g., $x_1 = 0$, $x_2 = 1$, and $x_3 = 0$. Such an assignment will either makes a clause *satisfy* or *not satisfy*. For example the clause $x_1 \vee x_2 \vee \bar{x}_3$ with assignment $x_1 = 0$, $x_2 = 1$, and $x_3 = 0$ evaluates to 1 which is satisfied. A SAT formula is a conjunction of a set of clauses. For example, F which is defined as $F = (x_1 \vee x_2 \vee \bar{x}_3) \wedge (x_2 \vee \bar{x}_4) \wedge x_3$ is a SAT formula.

Definition 1 (SAT Problem) *Given a SAT formula F on the variables $\{x_1, \ldots, x_n\}$ determine the satisfiability of F, i.e., determine if there exists an assignment on $\{x_1, \ldots, x_n\}$ so that F is satisfied. If such an assignment exists we might also want to find an explicit instance.*

Now the question is that why this is an interesting problem. In fact many real-world problems map naturally into a SAT problem. For example we can mention to problems of circuit design, compiler optimization, program verification, and scheduling.

The bad news is that Cook proved in 1973 that it is unlikely that there exists an algorithm which solves all instances of this problem in polynomial time (in n). More precisely, the problem is NP-complete.

We say that a formula F is k-SAT, $k \in \mathbb{N}$, if every clause involves exactly k literals. E.g., $(x_1 \vee x_2 \vee \bar{x}_3) \wedge (x_2 \vee x_3 \vee \bar{x}_4)$ is a 3-SAT formula. Then the following facts are known:

- 2-SAT formulas are easy to check for satisfiability. In the Homework 1 you will program a simple algorithm called unit-clause propagation. It solves a 2-SAT formula in at most $2n$ steps.
- The k-SAT problems are NP-complete for $k \geq 3$.

Graphical representation of SAT formulas (using factor graphs)

Given a SAT formula F, we associate to it a bipartite graph G. The vertices of the graph are $V \cup C$, where $V = \{x_1, \ldots, x_n\}$ are the Boolean variables and $C = \{c_1, \ldots, c_M\}$ are the M clauses. There is an edge between x_i and c_j if and only if x_i or $\bar{x}_i$ is contained in the clause c_j. Further we draw a "solid line" if c_j contains x_i and a "dashed line" if c_j contains $\bar{x}_i$.

Example 1 *As an example the graphical presentation of $F = (x_1 \vee x_2 \vee \bar{x}_3) \wedge (x_2 \vee x_3 \vee \bar{x}_4)$ is shown in Fig.1.* $\square$

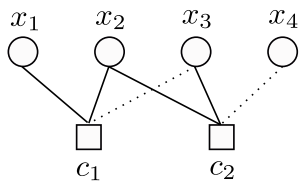

Figure 1: The factor graph corresponding to the SAT problem of Example 1.

Random k-SAT formulas

We will be interested in the behavior of random k-SAT formulas. So let us define an *ensemble* of such formulas. The ensemble $\mathcal{F}(n, k, M)$ is characterized by 3 parameters: k is the number of literals per clause, n is the number of Boolean variables, and M is the number of clauses.

How to sample from the $\mathcal{F}(n, k, M)$ ensemble

We define the $\mathcal{F}(n, k, M)$ ensemble by the way we sample from it. To this end, pick M clauses independently, where each clause is chosen uniformly at random from the $\binom{n}{k}2^k$ possible clauses. Then form F as the conjunction of these M clauses.

Now let us consider the following experiment. Fix $k \geq 3$ (e.g., $k = 3$) and sample from the $\mathcal{F}(n, k, M)$ ensemble. Now we ask the following question. Is such a formula satisfiable with high probability? It turns out that the most important parameter that affects the answer is $\alpha = \frac{M}{n}$.

As you observe in Fig. 2, as n becomes larger the transition of the probability of success becomes sharper and sharper. So it seems that it may exist a threshold behavior, i.e., there exist a real number α_k such that

$$\lim_{n \to \infty} \mathbb{P}\left[\mathcal{F}(n, k, M = \alpha n) \text{ is satisfied}\right] = \begin{cases} 0 & \alpha > \alpha_k, \\ 1 & \alpha < \alpha_k. \end{cases}$$

So we are interested in answering the following questions:

- Is there such a threshold behavior in this problem?
- If so, what is the threshold number α_k for a particular k?

- And finally, are there good algorithms to find satisfying assignments in the regime $\alpha < \alpha_k$?

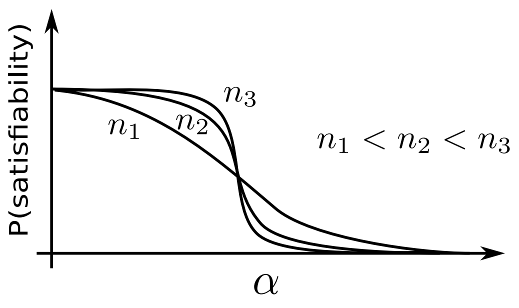

Figure 2: The probability that a formula generated from the random K-SAT ensemble is satisfied versus the clause density α.

3 Coding Problem

In this section we discuss about the other problem mentioned in §1. Let us start with defining what a binary code is.

Definition 2 *A binary block code $\mathcal{C}$ of length n is a collection of binary n-tuples, $\mathcal{C} = \{\underline{x}_1, \ldots, \underline{x}_\kappa\}$ where each $\underline{x}_i$ is called a codeword.*

We can narrow our attention to linear code which is defined as follows.

Definition 3 *A linear binary block code is a subspace of $\mathbb{F}_q^n$. Equivalently, for $\underline{x}_i$ and $\underline{x}_j$, $\underline{x}_i, \underline{x}_j \in \mathcal{C}$, then $\underline{x}_i - \underline{x}_j \in \mathcal{C}$. In particular $\underline{x}_i - \underline{x}_i = \mathbf{0} \in \mathcal{C}$. Since $\mathcal{C}$ is a subspace it has a dimension, call it k, $0 \leq k \leq n$. Hence, $\kappa = |\mathcal{C}| = 2^k$.*

For us it will be convenient to represent $\mathcal{C}$ as the kernel of a $(n-k) \times n$ binary matrix of rank $n-k$. This matrix is called the parity-check matrix and is usually denoted by H. So equivalently, we may write

$$\mathcal{C} = \left\{ \underline{x} \in \{0,1\}^n : \; Hx^\top = \mathbf{0}^\top \right\}.$$

The factor graph associated to the parity-check matrix H (of a code $\mathcal{C}$)

Assume that we have a linear code $\mathcal{C}$ defined by the $(n-k) \times n$ binary parity-check matrix H. We can associate to H the following bipartite graph G. The graph G has vertices $V \cup C$, where $V = \{x_1, \ldots, x_n\}$ is the set of n *variable* nodes corresponding to the n bits (and hence to the n columns of H), and $C = \{c_1, \ldots, c_{n-k}\}$ is the set of $n-k$ *check* nodes, each node corresponding to one row of H. There is an edge between i and c_j if and only if $H_{ji} = 1$.

Example 2 *Consider the following parity-check matrix*

$$H = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}.$$

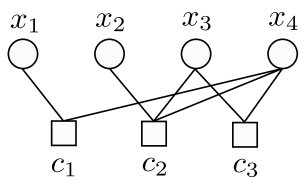

Figure 3: The factor graph corresponding to the parity-check matrix of Example 2.

The factor graph corresponding to H is shown in Fig. 3. $\square$

There are three related tasks to a coding problem which we will be going to discuss each briefly in the following.

- **Encoding:** Given $\mathcal{C}$, a code of dimension k, we can *encode* k bits of information by our choice of codewords among the 2^k possibilities. More precisely, we have an information word $\underline{u}$, $\underline{u} \in \{0,1\}^k$, and an encoding function g, $g : \{0,1\}^k \to \mathcal{C}$, which maps each information word into a codeword.

  Although, this function is of course crucial for real systems it only plays a minor role for our purpose. In fact the encoding process is not a difficult task.

- **Transmission (channel model):** We assume that we pick a codeword $\underline{x}$ uniformly at random from the code $\mathcal{C}$. We now *transmit* $\underline{x}$ over a *channel.* This channel models and abstracts the physical transition which takes place in a real system and typically leads to a *distortion* of the transmitted signal.
  Channel Model: Formally, the channel has the input alphabet $\mathcal{X} = \{0,1\}$ and an output alphabet $\mathcal{Y}$. We assume that the channel is memoryless and that there is no feedback. The channel is characterized by a transition probability $P(\underline{y}|\underline{x})$ where $\underline{y} \in \mathcal{Y}^n$ is the output and where

  $$P(\underline{y}|\underline{x}) = \prod_{i=1}^{n} p(y_i|x_i),$$

  since we assumed a memoryless channel with no feedback. As examples we can mention to the following channel: BEC[1], BSC[2], BAWGNC[3].

- **Decoding:** Given the output $\underline{y}$ we want to map it back to codeword $\underline{x}$; let $\hat{\underline{x}}(y)$ denotes this decoding function. There are many criteria to measure the performance of a decoding function but the most important ones are

  – the block error probability: $\mathbb{P}\left[\hat{\underline{x}}(y) \neq \underline{x}\right]$, and
  – the bit error probability: $\frac{1}{n}\sum \mathbb{P}\left[\hat{\underline{x}}(y)_i \neq x_i\right]$.

  Clearly, we would like to have low error probability using simple algorithms.

[1]Binary erasure channel.
[2]Binary symmetric channel.
[3]Binary additive white Gaussian noise channel.

The Gallagers' (l, r)-regular ensemble and the configuration model

As we have done for k-SAT problem, here, we define an ensemble of codes to study their performance under decoding algorithms. We focus on a specific ensemble of codes called (l, r)-regular Gallager ensemble which was introduced by Gallager in 1963.

The ensemble is characterized by the triple (n, l, r), where $n, l, r \in \mathbb{N}$, and also $n\frac{l}{r} \in \mathbb{N}$. The variable n is the length of the code, l is the variable node degree, and r is the check node degree.

To sample from the ensemble we proceed as follows. Pick n variable nodes and $n\frac{l}{r}$ check nodes. Each variable node has l *sockets* and each check node has r *sockets.* Number the ln variable sockets in an arbitrary but fixed way from 1 till nl. Do the same with the nl check node sockets. Pick a permutation π uniformly at random from the set Π of permutations on nl letters. For i from 1 till nl, insert an edge which connect variable node socket i to check node socket $\pi(i)$. If, after construction, we delete sockets then we get a bipartite graph, which we call the factor graph of the parity check matrix H.

The reason why we are interested in this ensemble is that the factor graph of a code in this ensemble has only ln edges. If, on the other hand, we had picked H randomly, we would expect $\frac{1}{2}n(n-k)$ non-zero entries, i.e., edges in the bipartite graph. We will see that a reasonable decoding algorithm consists of sending messages along the edges of the graph. So few edges means low complexity and, even more importantly, we will see that the algorithm works better if the graph is *sparse.*

In this context, there are some questions that we would like to investigate in this course. These are as follows:

- What are good and efficient decoding algorithms?
- If we pick a random such code from the ensemble, how well will it perform?
- In particular, is there going to be a threshold behavior so that for large instances the code *works* up to some noise level but *breaks down* above this level (see Fig. 4)?
- How can we compute this threshold and how does it compare to the Shanno capacity of the channel?

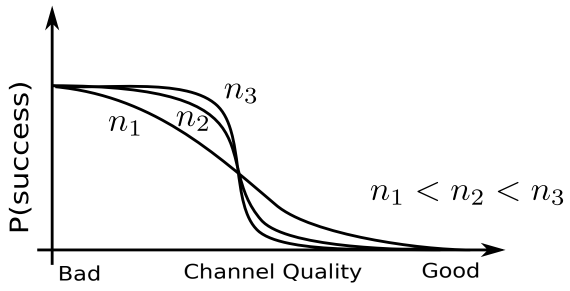

Figure 4: The probability of sucess of decoding the transmitted message versus the channel quality.