Statistical Physics for Communications, Signal Processing, and Computer Science

EPFL

Nicolas Macris and Rüdiger Urbanke

Contents

Foreword

Statistical physics, over more than a century, has developed powerful techniques to analyze systems consisting of many interacting "particles." In the last fifteen years, it has become increasingly clear that the very same techniques can be applied successfully to problems in engineering such communications, signal processing, or computer science.

Unfortunately there are several hurdles which one encounters when one tries to make use of these methods.

First, there is the language. Statistical mechanics has developed over the last 150 years with the aim of providing models and deriving predictions for various physical phenomenon, such as magnetism or the behavior of gases. This long history, together with the specific areas of their original application, has resulted in a rich language whose origins and meaning are not always clear to someone just starting in the field. It therefore takes a considerable effort to learn this language.

Second, except for extremely simple models, the "calculations" which are necessary are often long and daunting and frequently use little tricks and conventions somewhat outside the realm what one usually picks up in a calculus class. A good way of overcoming this difficulty is to start with a familiar example, casting it in terms of statistical physics notation, and by then going through some basic calculations.

Third, and connected to the second point, not all methods and tricks used in the calculations are mathematically rigorous. Some of the most powerful techniques, such as the cavity method, currently do not have a rigorous mathematical justification. In the "right hands" they can do miracles and give predictions which are currently not possible to derive with any classical method. But a newcomer to the field might quickly despair in trying to figure out what parts are mathematical rigorous and what parts are "most likely correct" but cannot currently be justified. Both worlds are valuable. The cavity or replica method give predictions which would be very difficult to guess. These predictions can then be used as a starting point for a rigorous proof. But it is important to cleanly separate the two worlds.

Our aim in writing these notes is not to give an exhaustive account of all there is to know about statistical mechanics ideas applied to engineering problems.

Indeed, several excellent books which take a much more in-depth look already exist. We in particular recommend [1, 2].

Our aim was to write the simplest non-trivial account of the most useful statistical mechanics methods so as to ease the transition for anyone interested in this strange put powerful world. Therefore, whenever we were faced with an option between completeness and simplicity, we chose simplicity. On purpose our language changes progressively throughout the text. Whereas at the beginning we try to avoid as much jargon as possible, we progressively start talking like a physicist. Most of the literature uses this language, so you better get used to it.

We decided to structure our notes around three important problems, namely error correcting codes, compressive sensing, and the random K-SAT problem. Although we will introduce basic versions of each of these problems, we only introduce what is necessary for our purpose. It goes without saying that there are myriad of versions and extensions, none of which we discuss. In fact, we hope that the reader is already somewhat familiar with these topics and accepts that these are important problems worth while studying. Using the basic versions of these problems we explain how they can be cast in a statistical physics framework and how standard concepts and techniques from statistical physics can be used to study these problems. This allows us to introduce the necessary terminology step by step, just when it is needed.

The notes are further partitioned into three parts. In the first part, comprised of Chapters 1-4, we introduce the problems, some of the language, and we rewrite these problems in the language of statistical physics. In the first chapter of the second part, namely Chapter 5, we then introduce the main protagonist, a message-passing algorithm which is also know as the *belief-propagation* algorithm. The remaining chapters of the second part, namely Chapters 6-12, contain the analysis of the performance of our three problems under this low-complexity algorithm. We will see that, in many cases, even this simple combination yields excellent performance. Finally, in the third part, consisting of Chapters 15-17, we get to the perhaps most surprising part of our story. Our aim will be to study the fundamental behavior of these three problems without the restriction to low complexity algorithms. I.e., how well would these systems work under optimal processing. The surprise is that the same quantities which appeared in our study of low-complexity suboptimal message-passing algorithms will play center stage also for this seemingly completely unrelated question.

Although we follow essentially the same pattern for each of the three problems, we will see that they are not all equally difficult.

Error correcting coding is perhaps easiest, and in principle most of the question one might be interested in can be answered rigorously. In this case we are dealing with large graphically models which are locally "tree like." It is therefore perhaps not so surprising that message-passing algorithms work well in this setting and that the performance can be analyzed.

Compressive sensing follows a similar pattern but introduces a few more wrinkles. In particular, the story of compressive sensing is leading to the so-called

AMP algorithm. The surprising fact here is that message-passing works very well, and that its performance can be predicted, despite that the relevant graphical model is not sparse at all but rather is a complete tree. The key observation is that every single edge contributes very little to the global performance. AMP can still be analyzed rigorously but the required computations are quite lengthy. We will give an outline of the whole story, but we will not discuss every single step in detail. Once the basic idea is clear, the interested reader should be able to fill in missing details by studying the pointers to the literature.

The hardest problem is without doubt the random K-SAT problem. We will only be able to present a partial picture. Many interesting and very basic questions remain open.

Many people have helped us in creating these notes. In the Spring of 2011 we gave a series of lectures on these topics at EPFL to mostly a graduate student population. We would like to thank Marc Vuffray, Mahdi Jafari, Amin Karbasi, Masoud Alipour, Marc Desgroseilliers, Vahid Aref, Andrei Giurgiui, Amir Hesam Salavati for typing up initial notes for some lectures. In addition we would like to thank Mike Bardet who typed up further material as well as Hamed Hassani who has since contributed material to several of the chapters.

Nicolas Macris, Lausanne, 2013
Rüdiger Urbanke

Part I

Models and their Statistical Physics Formulations

1 Models and Questions: Coding, Compressive Sensing, and Satisfiability

We start by introducing three problems: error correcting *coding*, *compressive sensing*, as well as *constraint satisfaction*. Although these three problems are quite different, we will see that essentially the same tools from statistical physics can be used to gain insight into their behavior as well as to make quantitative predictions. These three problems will serve as our running examples.

1.1 Coding

Error correcting codes

Codes are used in order to reliably transmit information across a noisy channel. Let us start with a basic definition. A *binary block code* $\mathcal{C}$ of length n is a collection of binary n-tuples, $\mathcal{C} = \{\underline{x}^{(1)}, \ldots, \underline{x}^{(\mathcal{M})}\}$, where $\underline{x}^{(i)}$, $1 \leq i \leq \mathcal{M}$, is called a codeword, and where the components of each codeword are elements of $\mathbb{F}_2 = (\{0,1\}, \oplus, \times)$, the binary field. The totale number of codewords is $|\mathcal{C}| = \mathcal{M}$ and the *rate* of the code is defined as $\frac{\log_2 |\mathcal{C}|}{n}$.

We will soon talk about various channel models, i.e., various mathematical models which describe how information is "perturbed" during the transmission process. In this respect it is good to know that for a large class of such models we can achieve optimal performance (in terms of the rate we can reliably transmit) by limiting ourselves to a simple class of codes, called linear codes.

A *linear binary block code* is a subspace of $\mathbb{F}_2^n$, the vector space of dimension n over the field $\mathbb{F}_2$. Equivalently, a binary block code $\mathcal{C}$ is linear iff for any two codewords $\underline{x}^{(i)}$ and $\underline{x}^{(j)}$, $\underline{x}^{(i)} - \underline{x}^{(j)} \in \mathcal{C}$. 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 $|\mathcal{C}| = 2^k$, and the rate of $\mathcal{C}$ is equal to $\frac{k}{n}$.

All codes which we consider in this course are binary and linear. Therefore, in the sequel we sometimes omit these qualifiers. It will be convenient to represent a linear binary code $\mathcal{C}$ of length n and dimension k as the kernel (or null space) of an $(n-k) \times n$ binary matrix of rank $n-k$. Such a matrix is called a *parity-check* matrix and is usually denoted by H. Every binary linear code has such a representation. So equivalently, we may write

$$\mathcal{C} = \left\{\underline{x} \in \mathbb{F}_2^n : H\underline{x}^\top = \mathbf{0}^\top\right\}$$

for some suitably chosen matrix H. The proof that at least one such matrix exists is the topic of an exercise.

A few remarks might be in order. First, once we have convinced ourselves that there is at least one such matrix, it is easy to see that there are exponentially many (in $n-k$) such matrices since elementary row operations do not change the row space and hence the code defined by the matrix. All these matrices define the same code, and are equivalent in this sense. But the representation of the code in terms of a bipartite graph, which we will introduce shortly, and the related message-passing algorithm, do depend on the specific matrix we choose and so our choice of matrix is important.

Second, and somewhat connected to the first point, rather than first defining a code $\mathcal{C}$ and then finding a suitable parity-check matrix H, we typically specify directly the matrix H and hence indirectly the code $\mathcal{C}$.

It can then happen that this matrix does not have full row rank, i.e., that its rank is strictly less than $n-k$. What this means is that the code $\mathcal{C}$ contains more codewords than 2^k. Since this will happen rarely, and since having more codewords than planned is in fact a good thing, we will ignore this possibility and only count on having 2^k codewords at our disposal.

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

Assume that we have a 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 where $C = \{c_1, \ldots, c_{n-k}\}$ is the set of $n-k$ *check* nodes, each node corresponding to one row of H. There is an edge between x_i and c_j if and only if $H_{ji} = 1$.

EXAMPLE 1 (Factor Graph) Consider the following parity-check matrix,

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

The factor graph corresponding to H is shown in Fig. 1.1. □

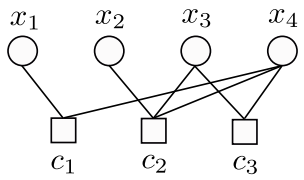

Figure 1.1 The factor graph corresponding to the parity-check matrix of Example 1.

Gallager's ensemble and the configuration model

A common theme in these notes is that instead of studying specific instances of a problem we define an *ensemble* of instances i.e., a set of instances endowed with a probability distribution. We then study the average behavior of this ensemble, and once the average is determined, we know that there must be at least one element of the ensemble with a performance at least as good as this average. In fact, in many cases, with a little extra effort one can often show that most elements in the ensemble behave almost as good as the ensemble average.

For coding, we focus on a specific ensemble of codes called the (d_v, d_c)-regular *Gallager* ensemble introduced by Gallager in 1961, [3, 4]. Rather than specifying the codes directly we specify their factor graphs. The ensemble is characterized by the triple of integers (n, d_v, d_c), such that $m = n\frac{d_v}{d_c}$ is also an integer. The parameter n is the length of the code, d_v is the variable node degree, and d_c is the check node degree.

To precisely describe the ensemble we explain how to sample from it. Pick n variable nodes and $n\frac{d_v}{d_c}$ check nodes. Each variable node has d_v *sockets* and each check node has r *sockets*. Number the $d_v n$ variable sockets in an arbitrary but fixed way from 1 till $d_v n$. Do the same with the $d_v n$ check node sockets. Pick a permutation π uniformly at random from the set of permutations on $d_v n$ letters. For $s \in \{1, \dots d_v n\}$ insert an edge which connects variable node socket s to check node socket $\pi(s) \in \{1, \dots d_v n\}$.

If, after construction, we delete sockets (and retain the connections between variable and check nodes) then we get a bipartite graph which is the factor graph representing our code. To this bipartite graph we can of course associate a parity-check matrix H. But note that in this model there can be multiple edges between nodes. A moments thought shows that the parity-check matrix H has a 1 at row i and column j if there are an odd number of connections between variable i and constraint j. Otherwise it has a 0 at this position. In practice multiple connections are not desirable and more sophisticated graph generation algorithms are employed. But for our purpose the typically small number of multiple connections will not play a role. In particular, it does not play a role if we are interested in the behavior of such codes for very large instances.

The above way of specifying the ensemble is inspired by the configuration model of random graphs, see [5]. This is why we call it the *configuration* model. This particular ensemble is a special case of what is called a *low-density parity-check* (LDPC) ensemble. This name is easily explained. The ensemble is *low-density* since the number of edges grows linearly in the block length. This is distinct from what is typically called the Fano random ensemble where each entry of the parity-check matrix is chosen uniformly at random from $\{0, 1\}$, so that the number of edges grows like the square of the block length. It is further a parity-check ensemble since it is defined by describing the parity-check matrix. 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*.

For many real systems, LDPC codes are the codes of choice. They have a very good trade-off between complexity and performance and they are well suited for implementations. "Real" LDPC codes are often further optimized. For example, instead of using regular degrees we might want to choose nodes of different degrees and the connections are often chosen with care in order to minimize complexity and to maximize performance. We will ignore these refinements in the sequel. The most important trade-offs are already apparent for the relatively simple regular Gallager ensemble.

Encoding, Transmission, and Decoding

The three operations involved in the coding problem are *encoding*, *transmission over a channel*, and *decoding*. Let us briefly discuss each of them.

Encoding: Given $\mathcal{C}$, a binary linear block code of dimension k, we can *encode* k bits of information by our choice of codeword, i.e., by choosing one out of the 2^k possibilities. More precisely, we have an information word $\underline{u}$, $\underline{u} \in \mathbb{F}_2^k$, and an encoding function g, $g : \mathbb{F}_2^k \to \mathcal{C}$, which maps each information word into a codeword.

Although this function is of crucial importance for real systems, it only plays a minor role for our purpose. This is true since, as we will discuss in more detail later on, for "typical" channels, by symmetry the performance of the system is independent of the transmitted codeword. We therefore typically assume that the all-zero codeword (which is always contained in a binary linear code) was transmitted. Also, in terms of complexity, the encoding operation is not a difficult task. One possible option is to write the linear binary code $\mathcal{C}$ in the form $\mathcal{C} = \{G\underline{u} : \underline{u} \in \mathbb{F}_2^k\}$, where G is the so-called *generator* matrix and where $\underline{u}$ is a binary column vector of length k which contains the information bits. In this form, encoding corresponds to a multiplication of a vector of length k with a $n \times k$ binary matrix and can hence be implemented in $O(k \times n)$ binary operations. In practice the code is often chosen to have some additional structure so that this operation can even be performed in $O(n)$ operations. We will hence ignore the issue of encoding in the sequel.

Transmission over a Channel: 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*". The actual channel is a physical device which takes bits as inputs, converts them into a physical quantity, such as an electric or optical signal, transmits this signal over a suitable medium, such as a cable or optical fiber, and then converts the physical signal back into a number which we can processed, perhaps equal to a voltage which is measured or the number of photons which were detected. Of course, during the transmission the signal itself is distorted. This distortion is either due

to imperfections of the system or due to unpredictable processes such as thermal noise. Instead of considering this potentially very complicated process we use a typically simple mathematical model which describes the end-to-end effect of all these physical processes on the signal. We call this model the "channel model."

Channel Model: Formally, the channel has the input alphabet $\mathcal{X} = \{0, 1\}$ and an output alphabet $\mathcal{Y}$. E.g., two common cases are $\mathcal{Y} = \{0, 1\}$ and $\mathcal{Y} = \mathbb{R}$. We assume that the channel is *memoryless*, which means that it acts on each bit independently. We further assume that there is no *feedback* from the output of the channel back to the input. In this case the channel is uniquely characterized by a transition probability $p(\underline{y} \mid \underline{x})$ where $\underline{y} \in \mathcal{Y}^n$ is the output and where

$$p(\underline{y} \mid \underline{x}) = \prod_{i=1}^{n} p(y_i \mid x_i). \tag{1.1}$$

Note that we get this product form from the assumptions that the channel is memoryless (acts bit-wise) and that we have no feedback.

The following three channels are the most important examples, both from a theoretical perspective, but also because they form the basis of real-world channels: These are the *binary erasure channel* (BEC), the *binary symmetric* channel (BSC) and the *binary additive white Gaussian noise channel* (BAWGNC).

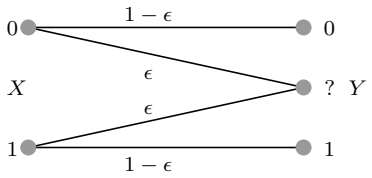

Figure 1.2 Binary erasure and symmteric channels with parameter ϵ.

BEC. The BEC is a very special channel with $\mathcal{Y} = \{0, ?, 1\}$. As depicted in Fig. 1.2, the transmitted bit is either correctly received at the channel output with probability $1 - \epsilon$ or erased by the channel with probability ϵ and thus, nothing is received at the channel output. The erased bits are denoted by "?". For example, if $x = 1$ is transmitted in the BEC, then the set of possible channel observation is $\{1, ?\}$. we may write somewhat formally for the transition probability $p(y|x) = (1 - \epsilon)\delta(y - x) + \epsilon\delta(y - ?)$.

BSC. The output of the BESC is binary $\mathcal{Y} = \{0, 1\}$. As seen on Fig. 1.2 the bit is transmitted correctly with probability $1 - \epsilon$ or flipped with probability ϵ. The transition probability is $p(y|x) = (1 - \epsilon)\delta(y - x) + \epsilon\delta(y - (1 - x))$.

BAWGNC. The output is a real number $\mathcal{Y} = \mathbb{R}$. When $x \in \{0, 1\}$ is sent the received signal is $y = x + z$ with z a Gaussian random number with zero mean and variance σ^2. With these conventions the "signal to noise ratio" is σ^{-2} and the transition probability $p(y|x) = (\sqrt{2\pi}\sigma)^{-1} e^{-\frac{(y-x)^2}{2\sigma^2}}$.

One might wonder if these three simple models even scratch the surface of the rich class of channels that one would assume we encounter in practice. Fortunately, the answer is *yes*. The branch of *communications theory* has built up a rich theory of how more complicated scenarios can be dealt with assuming that we know how to deal with these three simple models.

Decoding: Given the output $\underline{y}$ we want to map it back to a codeword $\underline{x}$. Let $\hat{x}(\underline{y})$ denote the function which corresponds to this *decoding* operation. What decoding function shall we use? One option is to first pick a suitable criterion by which we can measure the performance of a particular decoding function and then to find decoding functions which optimize this criterion. The most common such criteria are the *block error probability* $\mathbb{P}\left[\underline{\hat{x}}(y) \neq \underline{x}\right]$, and the *bit error probability* $\frac{1}{n}\sum_{i=1}^{n} \mathbb{P}\left[\underline{\hat{x}}(y)_i \neq x_i\right]$. We will come back in Chapter 3 to the precise definition of these error probabilities.

In practice, due to complexity constraints, it is typically not possible to implement an optimal decoding function but we have to be content with a low-complexity alternative. Of course, the closer we can pick it to optimal the better.

Shannon Capacity

So far we have defined codes, we have discussed the encoding problem, the process of transmission, the decoding problem, and the two most standard criteria to judge the performance of a particular decoder, namely the block and the bit error probability.

It is now natural to ask what is the maximum rate at which we can hope to transmit reliably, assuming that we pick the best possible codes and the best possible decoder. Reliably here means that we can make the block or bit probability of error as small as we desire. In fact, it turns out that the answer is the same wheter we use the block error probability or the bit error probability.

In 1948 Shannon gave the answer and he called this maximum rate the *capacity* of the channel. For binary-input memoryless output-symmetric channels the capacity has a very simple form. If the input alpahbet is binary and the output alphabet discrete, and if $p(y \mid x)$, $x \in \mathcal{X}$ and $y \in \mathcal{Y}$, denotes the transition probabilities, then the capacity of the associated channel can be expressed (in bits per channel use) as

$$H(p(\cdot)) - H(p(\cdot \mid x = 0)) \tag{1.2}$$

where $H(q(\cdot))$ denotes the entropy associated to a discrete distribution $q(y)$,

$y \in \mathcal{Y}$. By definition we have

$$H(q(\cdot)) = -\sum_{y \in \mathcal{Y}} q(y) \log_2 q(y). \tag{1.3}$$

Let us illustrate Shannon's formula for the BEC(ϵ). For $q(y) = p(y \mid x = 0)$ we have $q(0) = p(y = 0 \mid x = 0) = 1 - \epsilon$, $q(1) = p(y = 1 \mid x = 0) = 0$, and $q(?) = p(y = ? \mid x = 0) = \epsilon$. Further, for $q(y) = p(y) = \frac{1}{2}p(y \mid x = 0) + \frac{1}{2}p(y \mid x = 1)$ we have $p(0) = p(1) = \frac{1}{2}(1 - \epsilon)$ and $p(?) = \epsilon$. Hence, $H(p(\cdot)) = 1 - \epsilon + h_2(\epsilon)$ and $H(p(\cdot \mid x = 0)) = h_2(\epsilon)$, where $h_2(\epsilon) = -\epsilon \log_2 \epsilon - (1 - \epsilon) \log_2(1 - \epsilon)$ is the so called binary entropy function. We conclude that the capacity of the BEC(ϵ) is equal to $1 - \epsilon$. The capacities of the BSC and BAWGNC are computed similarly (see exercises).

That the capacity is at most $1 - \epsilon$ for the BEC is intuitive. For large block-lengths with high probability the fraction of non-erased positions is very close to $1 - \epsilon$. So even if we knew a priori which positions will be erased and which will be left untouched, we could not hope to transmit more than $n(1 - \epsilon)$ bits over such a channel. What is perhaps a little bit surprising is that this quantity is achievable, i.e., that we do not need to know a priori what positions will be erased and still can transmit reliably at this rate.

Questions

Now where we know the basic problem and have discussed the ultimate limit of what we can hope to achieve, the following questions seem natural to investigate.

- 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 as it is indicated schematically in Fig. 1.3? How does this threshold depend on the decoding algorithm?
- Assuming that there is a threshold behavior, how can we compute the thresholds?
- How do these thresholds compare to the Shannon threshold?

We will be able to derive a fairly complete set of answers to all of the above questions.

1.2 Compressive sensing

Basic problem

Here is the perhaps the simplest version of compressive sensing. Let $\underline{x}^{\text{in}} \in \mathbb{R}^n$ representing an "input signal" that we want to capture. We assume that the

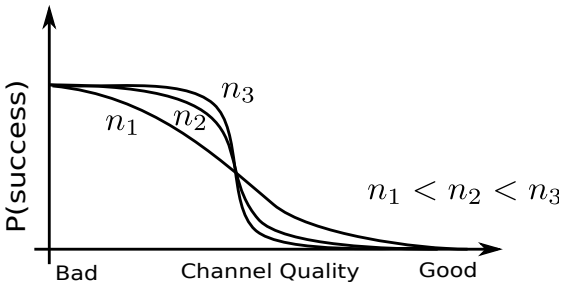

Figure 1.3 The probability of decoding error for a transmitted message versus the channel quality. As the blocklength of the code gets larger, we expect to see a sharper and sharper transition between range of the channel parameters where the system "works" and where it "breaks down."

number of non-zero components $\|\underline{x}^{\text{in}}\|_0 = |\{i|x_i^{\text{in}} \neq 0, i = 1, \ldots, n\}| = k$ of the signal is only a fraction of n; so $k = \kappa n$ with $\kappa < 1$ (and usually much smaller than one). The signal is captured thanks to an $m \times n$ "measurement matrix" A with real entries, $1 \leq m < n$. We set $m = \mu n$ with $\mu < 1$. Let $\underline{y} \in \mathbb{R}^m$ be given by $\underline{y} = A\underline{x}^{\text{in}}$. We think of $\underline{y}$ as the result of m linear measurements, one corresponding to each row of A. Our basic aim is to reconstruct the k-sparse signal $\underline{x}^{\text{in}}$ from the least possible measurements $\underline{y}$.

We know that at least one solution exists, namely $\underline{x}^{\text{in}}$, because the measurements $\underline{y}$ have been produced by this input signal. But since $m < n$, and in fact m is typically *much smaller*, we cannot simply solve the undetermined linear system of equations since the solution will not be unique. But we know in addition that $\underline{x}$ is k-sparse, i.e. has only k non-zero entries entries with $k < n$, (but we do not know which of these entries are non-zero). Therefore, we determine if the set of possible signals, namely

$$\{\underline{x} : A\underline{x} = \underline{y} \text{ and } \|\underline{x}\|_0 = k\}. \tag{1.4}$$

has cardinality one. If this is the case we may in principle be able to reconstruct our signal unambiguously.

One way to ensure the unicity of the solution is to take a measurement matrix A satisfying a *Restricted Isometry Property*. We say that A satisfies the RIP$(2k, \delta)$ condition if one can find $0 \leq \delta < 1$ such that

$$(1 - \delta)\|\underline{x}\|_2 \leq \|A\underline{x}\|_2 \leq (1 + \delta)\|\underline{x}\|_2, \quad \text{for all } 2k\text{-sparse vectors } \underline{x} \in \mathbb{R}^n. \tag{1.5}$$

It is not difficult to see that when this condition is met, then (1.4) has a *unique* solution given by

$$\hat{\underline{x}}_0(y) = \text{argmin}_{\underline{x}:A\underline{x}=y}\|\underline{x}\|_0. \tag{1.6}$$

Indeed, first notice that evidently $A\hat{\underline{x}}_0(y) = y$ so we only have to prove unicity. Suppose $\underline{x}'$ is another solution of (1.4). Then, since both $\underline{x}'$ and $\underline{x}_0(y)$ are k-sparse, their difference is $2k$-sparse. The left hand inequality of the RIP$(2k, \delta)$

condition states $(1-\delta)\|\underline{x}' - \hat{\underline{x}}_0(y)\|_2 \leq \|A\underline{x}' - A\hat{\underline{x}}_0(y)\|_2 = \|\underline{y} - \underline{y}\|_2 = 0$, which of course implies $\underline{x}' = \hat{\underline{x}}_0(y)$.

Solving the optimization problem (1.6) essentially requires an exhaustive search over $\binom{n}{k}$ possible supports of the sparse vectors, which is intractable in practice. One avenue for simplifying this problem is to replace the "ℓ_0 norm" in (1.6) with the ℓ_1 norm. In other words we solve the convex optimization problem,

$$\hat{\underline{x}}_1(y) = \text{argmin}_{\underline{x}:A\underline{x}=y}\|\underline{x}\|_1. \tag{1.7}$$

A fundamental theorem of Candes and Tao states that one can find δ', $0 < \delta' < \delta$, such that if A satisfies RIP$(2k, \delta')$ the solution of this problem is unique and identical to (1.6), [**?**].

This result shows that, for suitable measurement matrices, the ℓ_0 and ℓ_1 optimization problems are equivalent. Thus it suffices to solve the ℓ_1 problem. We will not prove it here but only offer some intuition for it through a simple toy example. Suppose that $n = 2$, so $\underline{x} = (x_1, x_2)^T$, and that we perform a single measurement $y = a_1x_1 + a_2x_2$. This equation corresponds to the line on figure

FIGURE

Figure 1.4 The ℓ_p balls

1.4. We seek to find a point on this line, which minimizes $(x_1^p + x_2^p)^{1/p}$, $p \geq 0$ where the case $p = 0$ is to be understood as the number of non-zero components of (x_1, x_2). As shown on figure 1.4 the solution is found by "inflating" the "ℓ_p-balls" around the origin until the line is touched. It is clear that for a generic line the solution is the same for all $0 \leq p \leq 1$. Note also that for $0 \leq p \leq 1$ the solution only has a single non-zero component, so is "sparse". For $p > 1$ the solution changes with p and both components are non-zero. Note when $p = 1$ there are non-generic measurement matrices corresponding to lines parallel to the faces of the ℓ_1-ball for which the solution is not unique; but as discussed shortly such cases will not bother us because the matrices will be chosen at random.

But what matrices satisfy the RIP condition ? It should come as no surprise that a matrix satisfying teh RIP condition should have a number of lines m at least as large as k. In fact one can show that necessarily $m \geq C_\delta k \log \frac{n}{k}$ for a suitable constant $C_\delta > 0$ [**?**]. It is not easy to make deterministic constructions of "good" measurement matrices approaching such bounds. The same is true with other deterministic conditions yielding equivalence of the ℓ_0 and ℓ_1 optimization

problems. However the toy example suggests that in fact all we migth need are "random measurement matrix". This is indeed a fruitful idea, at least in the asymptotic setting $n, m \to +\infty$ with $\kappa = \frac{k}{n}, \mu = \frac{m}{n}$ fixed, very much in the spirit of random coding. This is the route we will follow.

Ensembles of Measurement Matrices

While deterministic constructions of matrices satisfying the RIP condition are difficult, they can be shown to exist thanks to the probabilistic method [**?**]. The $m \times n$ matrix A will be taken from *the Gaussian ensemble* where the matrix entries are independent identically distributed Gaussian variables of zero mean and variance $1/m$. This normalization is such that each column of A has an expected ℓ_2 norm of 1. As in coding we will consider the assymptotic regime $n, m, k \to +\infty$ with *sparsity parameter* $\kappa = \frac{k}{n}$ and *measurement fraction* $\mu = \frac{m}{n}$ fixed. One can then show that there exists positive numerical constants c_1, c_2 such that for $m \geq c_1 \delta^{-2} k \log(\frac{en}{k})$ matrices from this ensemble satisfy the RIP(k, δ) condition with overwhelming probability $1 - \exp(-c_2 \delta^2 m)$ where the constants c_1, c_2 are numerical constants. More general ensembles are also possible.

The ensemble formulation for the measurement matrices, may also be extended to the signal model. One of the simplest signal distributions assumes that the components x_i are independently identically distributed according to a law of the form

$$p_0(x) = (1 - \kappa)\delta(x) + \kappa \phi_0(x) \tag{1.8}$$

where $\phi_0(x)$ is a continuous probability density. Depending on the model or the application $\phi_0(x)$ is known or unknown. The most realistic assumption for applications is to consider that $\phi_0(x)$ is unknown, and in that case we call $\mathcal{S}_\kappa$ this class of signals.

Noisy measurements and LASSO

A somewhat more realistic version of the measurement model is

$$\underline{y} = A\underline{x} + \underline{z},$$

where $\underline{z}$ is a noise vector, typically assumed to consist of m iid zero-mean Gaussian random variables with variance of σ^2. Again our aim is to reconstruct an k-sparse signal with as few measurements as possible. The matrix A is chosen from the random Gaussian ensemble and the signal from the class $\mathcal{F}_\kappa$.

If we ignored the sparsity constraint then it would be natural to pick the estimate $\hat{\underline{x}}(\underline{y})$ which solves the least-squares problem $\min_{\underline{x}} \|A\underline{x} - \underline{y}\|_2^2$. This problem is easily solved and the solution is well known $\hat{\underline{x}}(\underline{y}) = (A^T A)^{-1} A^T \underline{y}$. But in general this solution will not be k-sparse.

To enforce the sparsity constraint, we can add a second term to our objective

function, i.e., we can solve the following minimization problem,

$$\underline{\hat{x}}_0(\underline{y}) = \text{argmin}_{\underline{x}}(\|A\underline{x} - \underline{y}\|_2^2 + \lambda\|\underline{x}\|_0), \tag{1.9}$$

for a properly tuned parameter λ. Unfortunately this minimization problem is intractable, again because it requires an exhaustive search over the $\binom{n}{k}$ possible supports of the sparse vectors.

We saw in the noiseless case that replacing the "ℓ_0 norm" by the ℓ_1 norm is a fruitful idea. We follow the same route here and consider the following minimization problem

$$\underline{\hat{x}}_1(\underline{y}) = \text{argmin}_{\underline{x}}(\|A\underline{x} - \underline{y}\|_2^2 + \lambda\|\underline{x}\|_1). \tag{1.10}$$

This estimator is called the *Least absolute Shrinkage and Selectio Operator* (LASSO). Again λ has to be chosen appropriately. This estimator can in principle be calculated by standard convex optimizaton techniques, which is already a big improvement over exhaustive search.

Although the LASSO estimator is popular, its a priori justification is not so straightforward. Our discussion suggests that in the noiseless limit it reduces to the pure ℓ_1 estimator which we know gives for a certain range of parameters the correct solution of the ℓ_0 problem. This is one possible justification. Interestingly, the analysis of the LASSO in Chapter 10 the exact frontier for the ℓ_0-ℓ_1 equivalence in the (κ, μ) plane. This frontier is known as the Donoho-Tanner curve which they originally derived by completely different methods. In Chapter 3 we also discuss a somewhat more Bayesian justification of the LASSO in a setting where the signal distribution is not known, but only the parameter κ is assumed to be known. All this is ample justification for studying the LASSO in detail.

Graphical representation

As for coding one can set up a graphical representation for the measurement matrix. We associate to A a bipartite graph G with vertices $V \cup C$, where $V = \{x_1, \ldots, x_n\}$ is the set of *variable* nodes corresponding to the n signal components and $C = \{c_1, \ldots, c_m\}$ is the set of *check* nodes each node corresponding to a row (a measurement) of A. There is an edge between x_i and c_j if an only if $A_{ji} \neq 0$. For the random measurement matrices discussed above this will essentially always be the case and therefore the graph is simply the *complete bipartite* graph depicted on figure 1.5.

If one wishes one may atribute a "random weight" to the edges, but we will seldom need to do so. Therefore, unlike coding, here the graph is always the same. At this point this graphical construction may seem slightly trivial and arbitrary, but it will turn out to be a very useful way of thinking. The reason is that, much as in coding theory, we will develop iterative algorithms exchanging messages along the edges in order to reconstruct the signal. For example, this immediately suggests that the complexity of these algorithms scales like $O(n^2)$

FIGURE

Figure 1.5 The factor graph corresponding to the random gaussian 2×4 measurement matrix

because there are $nm = n^2\mu$ edges. Nevertheless each edge has a random weight of order $\pm 1/\sqrt{n}$ and this will allow us to reduce the complexity to $O(n)$.

Questions

Consider the regime where n tends to infinity and $\kappa = k/n$, $\mu = m/n$ constant.

- For given κ what fraction μ of measurements do we need so that with high probability we can recover $\underline{x}^{in}$ from the measurement $\underline{y}$ if we have no limitations on complexity?
- If we restrict ourselves to the low-complexity LASSO algorithm, how many measurements do we need then?
- Are there ways of designing compressive sensing schemes which achieve the theoretical limits under low-complexity algorithms?

1.3 Satisfiability

SAT problem

Suppose that we are given a set of n Boolean variables $\{x_1, \ldots, x_n\}$. Each variable x_i can take 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 operation "$\vee$" denotes the Boolean "or" operation. 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 make a clause to be *satisfied* or *not satisfied*. 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, i.e., the clause is satisfied. A SAT formula, call it F, is a conjunction of a set of clauses. For example, consider the SAT formula

$$F = (x_1 \vee x_2 \vee \bar{x}_3) \wedge (x_2 \vee \bar{x}_4) \wedge x_3.$$

where "$\wedge$" is the Boolean "and" operation.

The basic SAT problem is defined as follows. Given a SAT formula F, determine the satisfiability of F, i.e., determine if there exists an assignment on $\{x_1, \ldots, x_n\}$ so that F is satisfied. This is the SAT *decision* problem. If such an assignment exists we might also want to find an explicit solution.

Why on earth would anyone be interested in studying this question? Perhaps surprisingly, many real-world problems map naturally into a SAT problem. For example designing circuits, optimizing compilers, verifying programs, or scheduling can be phrased in this way. 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 SAT decision problem is NP-complete.

We say that a formula F is a K-SAT formula 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. The following facts are known. The 2-SAT decision problem is easily solved in a polynomial number of steps. Problem 1.6 discusses a simple algorithm called unit-clause propagation which solves a 2-SAT decision problem in at most $2n$ steps and produces a satisfying assignment if one exists. On the other hand for $K \geq 3$ the K-SAT decision problem is NP-complete.

Graphical representation of SAT formulas

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 2 (Factor Graph of SAT Formula) 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. 1.6. □

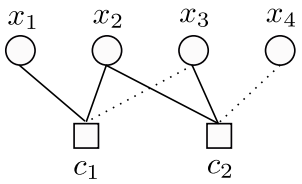

Figure 1.6 The factor graph corresponding to the SAT formula of Example 2.

Ensemble of random K-SAT Formulas

Just like in the coding and compressed sensing problems, rather than looking at individual SAT formulas, we will define an *ensemble* of such formulas and we will then study the probability that a formula from this ensemble is satisfiable. In particular, we will stick to the behavior of random K-SAT formulas.

The ensemble $\mathcal{F}(n, m, K)$ 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. Notice that with K variables we can form $\binom{n}{K}2^K$ clauses by taking K variables among $x_1, \ldots, x_n$ and then negating them or not. We define $\mathcal{F}(n, m, K)$ by showing how to sample from it. To this end, pick m clauses $c_1, \ldots, c_m$ 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. In other words, the ensemble $\mathcal{F}(n, m, K)$ is the uniform probability distribution over the set of all possible formulas F constructed out of n Boolean variables by choosing m clauses. The cardinality of this set is $\binom{\binom{n}{K}2^K}{m}$.

Threshold behavior

Now let us consider the following experiment. Fix $K \geq 2$ (e.g., $K = 3$) and draw a formula F from the $\mathcal{F}(n, m, K)$ ensemble. Is such a formula satisfiable with high probability? It turns out that the most important parameter that affects the answer is $\alpha = \frac{m}{n}$. This ratio is called the *clause density*. Like in coding and compressed sensing we are interested in the asymptotic regime where $n, m \to +\infty$ and α is fixed.

Fig. 1.7 shows the probability of satisfiability of F as a function of both n and α. As we see from this figure, as n becomes larger the transition of the probability of satisfiability becomes sharper and sharper. This is a strong indication that there exists a threshold behavior, i.e., there exists a real number $\alpha_s(K)$ such that

$$\lim_{n\to\infty} \mathbb{P}\left[F \text{ is satisfied}\right] = \begin{cases} 1, & \alpha < \alpha_s(K), \\ 0, & \alpha > \alpha_s(K). \end{cases} \tag{1.11}$$

Here $\mathbb{P}\left[-\right]$ is the uniform probability distribution of the ensemble $\mathcal{F}(n, m, K)$.

As the density α increases one has more and more clauses to satisfy, so it intuitively quite clear that the probability of satisfaction decreases as a function of α. However the existence of a sharp threshold is much less evident, let alone its computation. Such a threshold behavior was conjectured nearly two decades ago based on experiments []. For many years this was proved only for $K = 2$ for which $\alpha_s(2) = 1$. For $K \geq 3$ Friedgut proved that there exists a sequence $\alpha_s(K, n)$, $n \in \mathbb{N}$, such that for all $\epsilon > 0$

$$\lim_{n\to\infty} \mathbb{P}\left[F \text{ is satisfied}\right] = \begin{cases} 1, & \alpha < (1-\epsilon)\alpha_s(K, n), \\ 0, & \alpha > (1+\epsilon)\alpha_s(K, n). \end{cases} \tag{1.12}$$

This result leaves open the possibility that the sequence of thresholds $\alpha_s(K, n)$ does not converge to a definite value as $n \to +\infty$. The proof of a sharp threshold behavior (1.11) was proved recently in [] for K large enough (but finite), but for small K's (except $K = 2$) a proof is still a challenging problem.

The underpinnings of this proof for large K's rest on the statistical mechanics methods which also give the means to compute $\alpha_s(K)$ (for example it is known

that $\alpha_s(3) \approx 4.259$ to three decimal places). As we will see these methods yield much more information than just the threshold value. We will uncover various other threshold behaviors, related not only to the satisfiability of random formulas, but also to the nature of the solution space. Understanding the nature of these threshold behaviors in K-SAT is an order of magnitude more difficult than in coding theory and compressed sensing, and forms part of the more advanced material in chapters 18, 19.

Random max-K-SAT

In the K-SAT decision problem, one is given a formula and is asked to determine if this formula is satisfiable or not. An important variation on this theme is the *max-K-SAT* problem. In this problem one is interested in determining the *maximum possible number of satisfied clauses* where the maximum is taken over all possible 2^n assignments of variables $x_1, \ldots, x_n \in \{0,1\}^n$. Of course it is equivalent to determine the *minimum possible number of violated clauses* where the minimum is taken over all assignments of variables. In later chapters we will adopt this perspective which makes the contact with traditional statistical mechanics questions clearer.

We will be interested in the random version of max-K-SAT which we know formulate more precisely. Take a formula at random from the ensemble $\mathcal{F}(n, m, K)$. This formula contains m clauses labelled $c_1, \ldots, c_m$. If we let $\mathbb{1}_c(\underline{x})$ be the indicator function over assignments that satisfy clause c (i.e the function evaluates to 1 if $\underline{x}$ satisfies c and 0 if $\underline{x}$ does not satisfy c) then the maximum possible number of satisfied clauses is

$$\max_{\underline{x}} \sum_{i=1}^{m} \mathbb{1}_{c_i}(\underline{x})$$

In the random max-K-SAT problem we want to compute

$$\lim_{m\to+\infty} \frac{1}{m} \mathbb{E}\big[\max_{\underline{x}} \sum_{i=1}^{m} \mathbb{1}_{c_i}(\underline{x})\big] \tag{1.13}$$

where the expectation is taken over the ensemble $\mathcal{F}(n, m, K)$ (the existence of teh limit has been proven by methods that we will study in Chapter **??**). Equivalently we want to compute the average of the minimum possible number of violated clauses

$$e(\alpha) \equiv \lim_{m\to+\infty} \frac{1}{m} \mathbb{E}\big[\min_{\underline{x}} \sum_{i=1}^{m} (1 - \mathbb{1}_{c_i}(\underline{x}))\big] \tag{1.14}$$

We define the *max-K-sat threshold* as

$$\alpha_{s,\max}(K) = \sup\{\alpha | e(\alpha) = 0\} \tag{1.15}$$

We will give a non-rigorous computation of (1.14) and (1.15) in chapters 18,

19. In fact, the proof methods [] for the sharp threshold behavior (1.11) have their origin in such statistical mechanics computations.

Intuitively one expects that $\alpha_{s,\max}(K) = \alpha_s(K)$. It is clear that one must have $\alpha_s(K) \leq \alpha_{s,\max}(K)$. However the converse bound is not immediate because one could conceivably have a finite interval $]\alpha_s(K), \alpha_{s,\max}(K)[$ where $e(\alpha) = 0$ but at the same time a sublinear fraction of unsatisfied clauses. Nevertheless it is widely believed this does not happen and that $\alpha_s(K) = \alpha_{s,\max}(K)$. At least we know that this is true for $K = 2$ and for large enough (finite) K [] .

Questions

Here is a set of questions we are interested in:

- Does this problem exhibit a threshold behavior?
- If so, can we determine this threshold α_K?
- Are there low-complexity algorithms which are capable of finding satisfying assigments, assuming such assignments exist?
- If so, up to what clause density do they work with high probability?

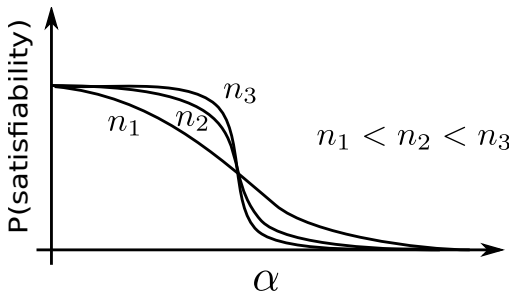

Figure 1.7 The probability that a formula generated from the random K-SAT ensemble is satisfied versus the clause density α.

Perhaps surprisingly, many of the above questions do not yet have a rigorous answer and the satisfiability problem is by far the hardest of our three examples. Nevertheless we will have non-trivial things to say about this problem and if one admits non-rigorous methods, the problem is fairly well understood.

1.4 Notes

Here we should put some further historical info as well as reference to the literature.

Problems

1.1 Capacity of the BSC and BAWGNC. Apply formula (1.2) to compute the Shannon capacity of the two channels.

1.2 Configuration Model. The aim of this problem is to write a program that can sample a random graph from the configuration model. Your program should take as input the parameters n, m, d_v, and d_c, it should then check that the input is valid, and finally return a bipartite graph according to the configuration model. Think about the data structure. If we run algorithms on such a graph it is necessary to loop over all nodes, refer to edges of each node, be able to address the neighbor of a node via a particular edge and store values associated to nodes and edges.

1.3 Norms and pseudo-norms. Let $\|\underline{x}\|_p = (\sum_{i=1}^n |x_i|^p)^{1/p}$ for $p > 0$. Let also $\|\underline{x}\|_0 = \sharp(\text{non zero } x_1, \ldots, x_n)$ and $\|\underline{x}\|_\infty = \max_i |x_i|$. Show first that $\|\underline{x}\|_0 = \lim_{p\to 0} \|\underline{x}\|_p$ and $\|\underline{x}\|_\infty = \lim_{p\to+\infty} \|\underline{x}\|_p$. Explain why $\|\cdot\|_p$ is a norm for $1 \leq p \leq +\infty$ and is *not* a norm for $0 \leq p < 1$ (this is why for $0 \leq p < 1$ we call it a pseudo-norm). *Hint*: refer to the figure 1.4.

1.4 Least square estimator. Show that the solution of the minimization problem (**??**) is $\hat{\underline{x}}(\underline{y}) = (A^T A)^{-1} A^T \underline{y}$.

1.5 Poisson Model. An important model of bipartite random graphs is the *Poisson model*. For example the random K-SAT problem is often formulated on this graph ensemble. Pick two integers, n and m. As before, there are n variable nodes and m check nodes. Further, let K be the degree of a check node. For each check node pick K variables uniformly at random either with or without repetition and connect this check node to these variable nodes. For each edge store in addition a binary value chosen according to a Bernoulli(1/2) random variable.

This is called the Poisson model because the node degree distribution on the variable nodes converges to a Poisson distribution for large n. This is also the case for the formulation in 1.3. The two formulation are equivalent in the asymptotic limit.

Write a program that takes n, m, K as input parameters and outputs a graph instance from the Poisson model. Again, think of the data structure.

1.6 Unit Clause Propagation for Random 3-SAT Instances. The aim of this problem is to test a simple algorithm for soving SAT instances. Generate random instances of the Poisson model. Pick $n = 10^5$ and let $K = 3$. Let α be a non-negative real number. It will be somewhere in the range $[0, 5]$. Let $m = \lfloor \alpha n \rfloor$. For a given α generate many random bipartite graphs according to the Poisson model. Interpret such bipartite graphs as random instances of a 3-

SAT problem. This means, the variables nodes are the Boolean variables and the check nodes represent each a clause involving 3 variables. Associate to each edge a Boolean variable indicating whether in this clause we have the variable itself or its negation.

For each instance you generate, try to find a satisfying assignment in the following greedy manner. This is called the *unit clause propagation* algorithm:

(i) If there is a check node in the graph of degree one (this corresponds to a *unit*-clause), then choose one among such check nodes uniformly at random. Set the variable to satisfy it. Remove the clause from the graph together with the connected variable and remove or shorten other clauses connected to this variable (if the variable satisfies other clauses they are removed while if not they are shortened).
(ii) If no such check exists, pick a variable node uniformly at random from the graph and sample a Bernoulli(1/2) random variable, call it X. Remove this variable node from the graph. For each edge emanating from the variable node do the following. If X agrees with the variable associated to this edge then remove not only the edge but the associated check node and all its outgoing edges. If not, then remove only the edge.

Continue the above procedure until there are no variable nodes left. If, at the end of the procedure, there are no check nodes left in the graph (by definition all variable nodes are gone) then we have found a satisfying assignment and we declare success. If not, then the algorithm failed, although the instance itself might very well be satisfiable.

Plot the empirical probability of success for this algorithm as a function of α. You should observe a threshold behavior. Roughly at what value of α does the probability of success change from close to 1 to close to 0?