University of Alberta

ON RANDOM FIELD CAPTCHA GENERATION

by

Fraser Newton

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Master of Science

in

Statistics

Department of Mathematical and Statistical Sciences

Fall 2012
Edmonton, Alberta

To Maureen.

Abstract

In this thesis, we develop a novel method of generating CAPTCHAs, which are used to protect online resources from abuse by computer agents. We view CAPTCHA generation as random field simulation and construct a CAPTCHA by evolving an initial state via resimulating pixels until the image becomes readable. We empirically demonstrate that this CAPTCHA is easy for humans to read but difficult for computer programs to crack.

We describe how to develop variants of this CAPTCHA; in particular, we implement and assess the utility of a grey-level variant. We establish a method of maximizing the effectiveness of a CAPTCHA variant, and perform analysis to determine which properties of the CAPTCHA most effectively differentiate humans and computer programs.

We extend the random field used in the CAPTCHA application to multiple dimensions in the context of graph theory, and describe the generic method of applying the random field to suitable problems.

Acknowledgements

First, I would like to thank my supervisor, Michael Kouritzin, for his contributions to as well as his encouragement of this work. I am also grateful to Biao Wu, who always took the time to discuss the random field described within and never hesitated to answer my many questions. In addition, I thank Byron Schmuland, Keumhee Carrière Chough, and Thomas Hillen for their thoughtful comments and questions on this work.

I thank the anonymous volunteers who partook in the readability study at `http://www.knwcaptcha.org`. Their feedback and the results collected through this study formed an early and integral foundation of this work.

I gratefully acknowledge the support of NSERC through the CGS-M, without which I would not have been able to pursue my interest in probability and statistics.

Finally, I thank my friends and family for believing that I could.

Table of Contents

List of Tables

List of Figures

Nomenclature

$\beta_{h,t}$	Site-site covariances, page 13
$\mathbf{X}$	State space, page 43
$\mathbf{X}_v$	State space at vertex v, page 65
$\mathcal{X}_A$	Product of state spaces in A, page 43
$\partial_\ell(s)$	ℓ-neighborhood, page 12
$\partial_v()$	Parents of vertex v in a directed graph, page 65
Π	Probability measure, page 13
π_h	Site marginals, page 13
$\rho(s,t)$	Euclidean distance, page 12
Θ	Space of CAPTCHA generation parameters, page 51
θ	CAPTCHA generation parameters, page 51
A_h	Known sites in the neighbourhood of h, page 43
$D = (V, A)$	Directed graph, page 64
$f(\theta)$	Cost function of CAPTCHA, page 52

$G = (V, E)$	Undirected graph, page 63
H	Unknown part of S, page 42
$h(\theta)$	Probability of human recognizing CAPTCHA, page 52
H^C	Known part of S, page 42
$N_G(v)$	Neighbourhood of vertex v, page 63
S	Sites, page 12
$t(\theta)$	Probability of OCR program recognizing CAPTCHA, page 52
X	Random field, page 43
pmf	Probability mass function, page 13

Chapter 1

Introduction

In this work, I present three papers prepared during my Masters of Science program at the University of Alberta. "On Random Field CAPTCHA Generation" Kouritzin *et al.* [3], accepted with mandatory minor revisions to IEEE Transactions on Image Processing on 2012-07-05, is reproduced in Chapter 2 and describes the problem of automatically differentiating computer programs from humans for the purposes of protecting online resources and its solution, CAPTCHAs. CAPTCHAs are typically scrambled or deformed word images which a user must correctly identify in order to access a resource such as web-mail or an online survey. We discuss the desired properties of CAPTCHAs, review prior solutions and their vulnerabilities, and introduce our own novel method of generating black and white CAPTCHAs via pixel-by-pixel simulation. Our method views CAPTCHA generation as random field generation, where each pixel corresponds to a discrete random variable, and the structure of the CAPTCHA is captured by the marginal probabilities and pairwise covariances between pixels; these parameters are estimated from randomly constructed instances of a given CAPTCHA. We empirically determine that our KNW-CAPTCHA is both highly readable to humans and difficult for computer programs to defeat, and empirically compare our own method to implementations by Google, YAHOO, and eBay. Furthermore, we describe how to implement diverse variants of our CAPTCHA in a straightforward manner and a method of selecting the best variant given a desired balance of readability and attack resistance.

An abridged and slightly modified[1] "On Grey Levels in Random CAPTCHA Generation" **Newton** *et al.* [4], published in Proceedings of SPIE Visual Information Processing XX, is given in Chapter 3 and builds off the work established in [3]. In particular, we examine the effectiveness of a grey-level variant of the KNW-CAPTCHA, where we define CAPTCHA effectiveness as a function of its human readability and attack resistance. We detail exactly how to incorporate the grey level into the parameter estimation, provide a cost function which incorporates human readability and attack resistance as well as a method of estimating it, and empirically determine the optimal CAPTCHA parameters for given balances between human readability and attack resistance. Finally, we perform logistic regression on the human and computer responses to the CAPTCHA challenges in order to determine the relationship and significance of the CAPTCHA parameters, as well as to identify which parameters are likely to aid in the effective differentiation of human and computer agents.

"The KNW Random Field" [5], a manuscript in final preparation which will be submitted to Computation Statistics & Data Analysis, is reproduced in part in Chapter 4. We provide a version of the algorithm applied in [3] and

[1]The analysis in Section 3.E has been revised and extended to include evaluation of effect size.

[4] generalized from two dimensions to multiple dimensions; specifically, we set up the problem as a graph, where vertices would correspond to pixels in the previous applications and edges between vertices correspond to the pairwise covariances. We prove that the resulting random field has the desired marginal probabilities and covariances and provide explanatory examples.

Chapter 2

On Random Field CAPTCHA Generation[1]

2.A Introduction

A CAPTCHA is a "Completely Automated Public Turing test to tell Computers and Humans Apart" von Ahn *et al.* [6], widely used to protect online resources from abuse by automated agents. Von Ahn *et al.* [7] suggests that hard artificial intelligence (AI) problems form the test basis and defines a (α, β, η)-CAPTCHA as a test that 1) can be solved by at least α proportion of humans (e.g., the English-reading adult portion) with a probability of success greater than β; 2) if a computer program can solve it with probability greater than η in fixed time, then the program can be used to solve the hard AI problem (see [7] for details). A common CAPTCHA is an image of (usually alphanumeric) characters that are easy to identify by English-reading

[1]A version of this chapter has been submitted for publication. *IEEE Transactions on Image Processing.*

humans yet translate into the hard AI problem of optical character recognition (OCR). Segmentation of characters within a word image is error prone [8], and continues to be difficult for contemporary OCR algorithms [9]. Therefore, segmentation should be hard to ensure an OCR-based CAPTCHA is resistant to computer programs.

Herein, we introduce a general method for generating “KNW-CAPTCHAs”, with the view that random CAPTCHA creation is really random field simulation. We simulate random fields with given pixel marginal probabilities and pixel-pixel correlations, which are estimated from a priori samples with random variations in the fonts and placement of letters. This can be thought of as a form of lossy compression: while the complete information is the joint distribution, we store only the marginal probabilities and covariances, from which a (possibly different) joint distribution can be reconstructed. However, we simulate directly from the marginal probabilities and covariances. A KNW-CAPTCHA is initialized as a random field, and the CAPTCHA is then generated via partial Gibbs re-sampling in order to provide enough information to make the test word human-recognizable, yet ensure that OCR remains hard. In contrast to other methods which apply deformations to an initial word image, the KNW-CAPTCHA is a partial evolution from OCR-disruptive noise towards a random word image.

For an effective (α, β, η)-CAPTCHA, β should be high and η should be low. The target population for our KNW-CAPTCHAs is English-readers with better than 20/60 vision (though we have little control over the participants in our readability studies). We establish high β via a readability study and endorse low η via experiments with modern OCR programs.

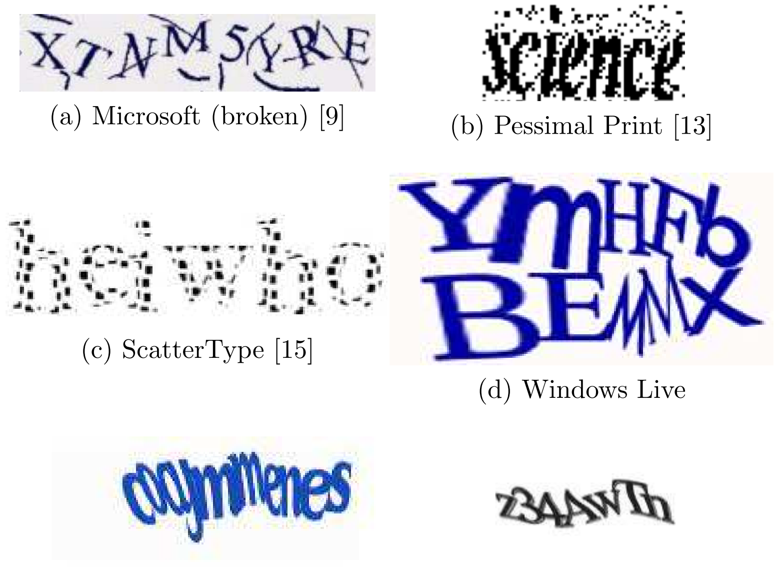
We begin with an overview of past and present text-based CAPTCHAs.

(While there are many alternatives to text-based CAPTCHAs, such as the image-based IMAGINATION [10], which requires users to annotate images, text-based CAPTCHAs continue to be the de facto standard in industry.) The early, now broken PayPal and the Microsoft CAPTCHAs discussed in [11] and [9], respectively, both relied on background noise and random character strings to resist automated attacks but did not employ character crowding, significant distortion, nor sophisticated random field techniques. The background noise (random arcs in [9] - see Figure 2.1) was trivial to remove due to its distinctiveness.

Mori *et al.* [2] successfully attack both EZ-Gimpy and Gimpy CAPTCHAs. EZ-Gimpy uses word images, and employs clutter and character distortion to defend against attacks. However, it does not employ character crowding. The authors of [2] make use of character shape contexts in order to obtain many candidate letter locations and exploit EZ-Gimpy's use of words. Gimpy's clutter is two distorted overlapping word images (chosen from a dictionary of 411). In a CAPTCHA challenge, five pairs of overlapping words are presented. In [2], the authors determine the opening and closing bigrams of each word and use this knowledge to prune the space of possible words. Further pruning is accomplished using word-sized shape contexts. *Moy et al.* [12] break EZ-Gimpy and Gimpy-r. Gimpy-r presents the user with four random, distorted character images from an alphabet of 19 letters against a cluttered background. It does not, however, use character crowding nor random field techniques to impede segmentation. The authors of [12] are able to remove the background clutter and segment the challenge into four character recognition problems, which are solved by determining which template character image requires the least distortion to match the observed character image. (Performance is further improved using additional steps.)

Pessimal Print (see Figure 2.1b), introduced in Coates *et al.* [13], simulates low-quality print images that challenge OCR. The CAPTCHA generation randomly selects a word, a font, and a set of image degradation parameters to thicken, crowd, fragment, and add noise to character images. 685 word images were generated; all were readable to the ten human volunteers, while almost all were unrecognizable to the Expervision TR, ABBYY FineReader, and IRIS Reader OCR programs. Furthermore, OCR performance was very sensitive to changes in the parameters.

Chew et al.'s [14] BaffleText CAPTCHA relies on a human's Gestalt perception, i.e., the ability to assemble the whole given fragments of an image. BaffleText generates pronounceable non-English random character strings, displayed in a randomly selected font and masked by random circles, squares, and ellipses using one of the pixel-wise boolean operations "or", "not and", or "exclusive or". Character strings are generated using a trigram Markov model to solve the small dictionary problem that can plague English word-based CAPTCHAs; random masks are used over simple additive pixel noise in order to exercise humans' Gestalt perception. Human readability results were collected from 33 volunteers on 1212 BaffleText images, with 79% success. Attack resistance is established by subjecting BaffleText images to the attack described in [2]. The attack succeeded on only 11% of the BaffleText images, lower than both Pessimal Print and EZ-Gimpy. The ScatterType CAPTCHA (see Figure 2.1c), introduced in Baird *et al.* [15], also relies on Gestalt perception. Pseudo-words are generated using an n-gram Markov model; then each character in the word is cut vertically and horizontally and the resulting fragments are displaced randomly.



(a) Microsoft (broken) [9]

(b) Pessimal Print [13]

(c) ScatterType [15]

(d) Windows Live

(e) Google

(f) Yahoo!

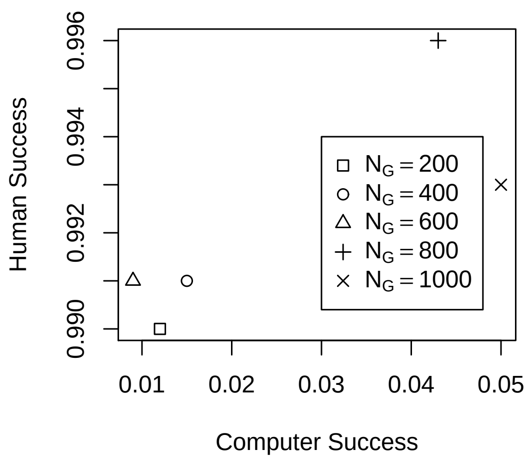
Figure 2.1: A few CAPTCHA Examples

Finally, we examine some popular CAPTCHAs in use today. The CAPTCHAs used by Google, Yahoo!, and Windows Live (see Figure 2.1) all share similar properties: a lack of background noise, distortion of character or word images, and extreme crowding of adjacent characters. Segmentation resistance is largely accomplished by character crowding, notably lacking from earlier, now broken CAPTCHAs such as the captchaservice.org CAPTCHAs in [16], the PayPal CAPTCHA in [11], the Microsoft CAPTCHA in [9], EZ-Gimpy in [2], and Gimpy-r in [12]. However, this extreme crowding also makes human-recognition a challenge. For example, is it obvious what the character string in the Google CAPTCHA is?

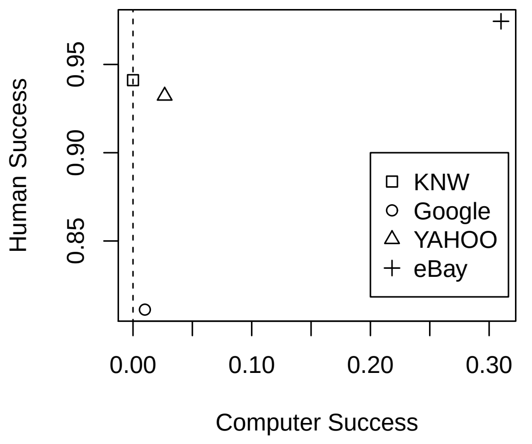
In contrast with the methods covered above, we view CAPTCHA generation as correlated random field simulation. Like Pessimal Print [13], our images provide partial, noisy information. We also leverage Gestalt perception to maintain a human-readable image, as in [14] and [15]. However, our use of randomness is far more fundamental and thereby far harder for computers to deal with than prior methods. We observe that the human readability of random CAPTCHA images is captured by the site, i.e. pixel, marginal probabilities and the site-to-nearby-site covariances; the actual joint distribution of the sites is not so important. Our method begins with a correlated random image that is evolved randomly a site at a time via Gibbs sampling until the random test word is human-readable. Our method of calculating each site's conditional probability mass function given the nearby sites that are either known or already simulated gives us exactly what is required for Gibbs sampling. The initial image can be a simple white background, any correlated random field, or, for strong segmentation resistance, a CAPTCHA generated by the ScatterType algorithm [15] with a different base word. Both the legibility and segmentation-resistance of our KNW-CAPTCHA depends on the number of iterations used in the Gibbs sampling step. The upshot is that we generate flexible, random CAPTCHAs automatically and efficiently and explain exactly how we do it.

In this work, we investigate two variants of the KNW-CAPTCHA: the KNW-CAPTCHA$_{\mathrm{E}}$, an easy variant generated without any background noise, and the KNW-CAPTCHA$_{\mathrm{H}}$, which is generated using character fragments as the background noise. The KNW-CAPTCHA$_{\mathrm{E}}$ is used to investigate how the generation parameters (especially the number of Gibbs iterations, N_G) affect the attack resistance of the resulting CAPTCHA. Figure 2.2a shows both the attack resistance and human readability of the KNW-CAPTCHA$_{\mathrm{E}}$ for various values of N_G, where computer success is the proportion of CAPTCHAs where either of the OCR programs Tesseract or ABBYY FineReader successfully recognized it, and human success is the proportion of CAPTCHAs where a human successfully recognized it. The KNW-CAPTCHA$_\mathrm{H}$ would be used in practice as the background noise provides additional security but the CAPTCHA remains highly readable to humans. Figure 2.2b compares the human readability and attack resistance of the KNW-CAPTCHA$_\mathrm{H}$ with several CAPTCHAs deployed by major corporations. As the correct answers for the comparison CAPTCHAs are unknown, we use optimistic solving accuracy (see Section 2.D.4) to determine human success; similarly, an OCR program is considered correct if it matches any of the human responses. These graphs clearly illustrate that both the KNW-CAPTCHA$_\mathrm{E}$ and KNW-CAPTCHA$_\mathrm{H}$ are highly readable and difficult to attack; even the KNW-CAPTCHA$_\mathrm{E}$ appears to have resistance to OCR comparable to or surpassing CAPTCHAs currently used by Google, YAHOO, and eBay. There were no computer successes against the KNW-CAPTCHA$_\mathrm{H}$, yet it obtained over 94% human success. (None of the other CAPTCHAs went unrecognized by OCR; only the eBay CAPTCHA bested the KNW-CAPTCHA$_\mathrm{H}$ in human success, but it also appears to be trivially broken.)

Our notation and random field algorithm are given in Section 2.B. Section 2.C details our CAPTCHA generation, and Section 2.D contains our results. We discuss alternative implementations of the KNW-CAPTCHA in Section 2.E.1. The mathematics behind the methodology in this paper will be published separately (see [5], [17]).



(a) KNW-CAPTCHA$_\mathrm{E}$ with various N_G



(b) KNW-CAPTCHA$_\mathrm{H}$, Google, YAHOO, and eBay

Figure 2.2: Human and Computer Success on Various CAPTCHAs

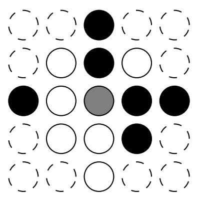


Figure 2.3: Simulation Example

2.B Notation, Background, and Probability Computation

We begin by giving the required mathematical background and the equation for the conditional probability of a pixel given the nearby pixels based on correlations and marginal probabilities. Our goal is to randomly turn a pixel on/off given an estimated set of parameters (the marginal probabilities and site-site covariances) and the values of nearby pixels. The parameters capture the fundamental properties of the challenge word and, as pixels are re-simulated, the random image approaches the desired image. Figure 2.3 illustrates this setup, where the grey node represents the pixel being simulated and the nodes with solid outlines represent the nearby already-simulated pixels.

We consider a rectangular image of $M \times N$ pixels at the sites $S = \{(i,j) : 1 \leq i \leq M, 1 \leq j \leq N\}$, let $\rho(s,t) = \sqrt{(i_2 - i_1)^2 + (j_2 - j_1)^2}$ be the Euclidean distance between $s = (i_1, j_1)$ and $t = (i_2, j_2)$, and define the neighborhoods of $s = (i,j) \in S$ with radius $\ell \in \mathbb{R}$ as the ℓ-neighborhood

$$\partial_\ell(s) = \{(u,v) \in S : 0 < \rho((i,j),(u,v)) \leq \ell\}.$$

Definition 2.B.1. A point $s = (i,j) \in A$ is *ℓ-connected within* set $A \subset S$ if $\partial_\ell(s) \cap A$ is not empty. A is *ℓ-connected* if for every proper subset $B \subset A$,

$\partial_\ell(B) \cap A$ is not empty.

We assume the desired *site marginals* $\{\pi_h\}$ satisfy $\pi_h(1) = 1 - \pi_h(-1) \in (0,1)$, $h \in S$ and $\{\beta_{h,t} : t \in \partial_\ell(h), h \in S\}$ are *site-site covariances.* Assume the numbers on the RHS of equation (2.1) (to follow) are in $[0,1]$ (conditions for this to be true are given in [17]). Then, there is a probability measure Π on $\{-1,1\}^{MN}$, where -1 corresponds to white and 1 corresponds to black, such that for each $h \in S$

$$\Pi(X_h = c) = \pi_h(c),\ \forall\ c \in \{-1,1\},\ \operatorname{cov}(X_h, X_t) = \beta_{h,t},\ \forall\ t \in \partial_l(h),$$

i.e., with correct marginals and covariances, and

$$\Pi(X_h = x_h |\, X_{\partial_\ell(h)} = x_{\partial_\ell(h)}) = \quad \pi_h(x_h) + \frac{\sum\limits_{t \in \partial_\ell(h)} x_h \beta_{h,t} x_t}{\frac{1}{4} 2^{|\partial_\ell(h)|+1} \Pi(X_{\partial_\ell(h)} = x_{\partial_\ell(h)})} \tag{2.1}$$

for each $x_h \in \{-1,1\}$ and $x_{\partial_\ell(h)} \in \{-1,1\}^{|\partial_\ell(h)|}$, where $|\cdot|$ denotes the cardinality of a set.

Now, we explain how we use the marginals and covariances to determine the conditional probabilities (2.1) for simulating a KNW-CAPTCHA. Suppose we have determined the site probability mass functions (pmf's) $\{\pi_h\}_{h \in S}$ and the covariances $\{\beta_{h,t} : h, t \in S \text{ and } \rho(h,t) \le \ell\}$ of sites within distance ℓ of each other for random instances of the challenge word. (This is dealt with below.) Then, we start with a random field designed to bait computers into the wrong conclusions. Finally, we resample using $\{\pi_h\}$ and $\{\beta_{h,t}\}$ together with (2.1) until the challenge word is just human-readable yet there is such correlated noise that automated agents are unable to recognize the text.

We resample using Gibbs-like sampling, where we condition only on a large area around a site instead of all sites. The algorithm will randomly select a site $h \in S$ to resample using (2.1) to ensure we keep the desired pmf's and covariances. The joint probability $\Pi(X_{\partial_\ell(h)} = x_{\partial_\ell(h)})$ in the denominator on the RHS of (2.1) can be computed easily in real time by caching and re-using results. Let $\{t_1, \ldots, t_{|\partial_\ell(h)|}\}$ be the sites in $\partial_\ell(h)$ and $B_k = \{t_1, ..., t_k\}$ for $k = 1, \ldots, |\partial_\ell(h)|$ and $B_0 = \emptyset$. Then, we compute $\Pi(X_{\partial_\ell(h)} = x_{\partial_\ell(h)})$ using the multiplication rule

$$\Pi(X_{\partial_\ell(h)} = x_{\partial_\ell(h)}) = \prod_{i=1}^{|\partial_\ell(h)|} \Pi(X_{t_i} = x_{t_i} | X_{B_{i-1}} = x_{B_{i-1}}). \tag{2.2}$$

$\Pi(X_{t_i} = x_{t_i} | X_{B_{i-1}} = x_{B_{i-1}})$, $i = 1, \ldots, |\partial_\ell(h)|$ can be computed directly using (2.1).

Next, based on the conditional probabilities computed using (2.1), we use the following straightforward simulation algorithm to simulate h with the appropriate marginals and covariances.

1. Compute $\Pi(X_h = c^u | X_{\partial_\ell(h)} = x_{\partial_\ell(h)})$ for $1 \leq u \leq d$, using (2.1).

2. Generate a $[0, 1]$-uniform random variable U. If

$$\sum_{u=1}^{w-1} \Pi(X_h = c^u | X_{\partial_\ell(h)} = x_{\partial_\ell(h)}) \leq U < \sum_{u=1}^{w} \Pi(X_h = c^u | X_{\partial_\ell(h)} = x_{\partial_\ell(h)})$$

   for some $1 \leq w \leq d$, then we set $X_h = c^w$, i.e., the realization of X_h is c^w.

2.C The KNW-CAPTCHA

We now present how to estimate the required parameters for a particular KNW-CAPTCHA and use those parameters to generate a novel random CAPTCHA in Sections 2.C.1 and 2.C.2, respectively.

2.C.1 Parameter Estimation

We begin by generating the data for the estimation process that consists of many independent instances of a particular word, where each instance varies randomly in many ways. The parameters learned from this data will represent the challenge word; by learning the parameters (site probabilities and site-to-nearby-site covariances) from this data, we can construct the conditional probabilities of the previous section and, thereby, do the Gibbs resampling portion of our CAPTCHA creation.

The algorithm for generating the data consists of selecting a word to serve as the KNW-CAPTCHA's correct response and then generating a number of random images representing this word by varying fonts and placement of characters in the word. The word images will be constructed by joining individual character images. Herein, we select a random word uniformly over a fixed dictionary of common English words with a length of at least three characters.

For each letter in the English alphabet and for each of 18 fonts, we generate character images denoted $\{f_{1,1}, \ldots, f_{1,26}, \ldots, f_{18,1}, \ldots, f_{18,26}\}$, i.e., $f_{i,j}$ is the character image of the j^{th} letter in the i^{th} font. To ensure that forming a word image by joining random character images results in consistent horizontal placement of individual character images, we work with character images that, for a given letter, all have the same width. To accomplish this, we generate trimmed or scaled character images for each letter as appropriate. Let N_j^f denote the maximum width of the bounding boxes over the character images $\{f_{1,j}, \dots, f_{18,j}\}$, where a bounding box is the smallest rectangle that encloses the character. For $i = 1, \dots, 18$, $j = 1, \dots, 26$,

- if j is one of the letters {i,j,l,r,t}, generate a new character image $f'_{i,j}$ by centering and trimming $f_{i,j}$ so that its width is N_j^f by removing columns outside the bounding box;

- otherwise, generate a new character image $f'_{i,j}$ by scaling $f_{i,j}$ so that $f'_{i,j}$'s bounding box has a width of N_j^f and removing all columns outside the bounding box.

The letters {i,j,l,r,t} were chosen for trimming instead of scaling since scaling some of their images results in very tall bounding boxes due to their highly variable character widths.

We then generate K images of the chosen character string with pixel state space $\{-1, 1\} = \{\text{white}, \text{black}\}$, and n_c is the number of characters in the character string using the following algorithm.

1. The horizontal distance between each adjacent character's bounding box is chosen using a random number selected uniformly over $\{1, 2, 3\}$. This is fixed for all K images.

2. The vertical displacements of characters are determined using the values $\{v_0, v_1, v_2, \dots\}$ of a reflecting random walk, moving upward or downward one with probability $\frac{1}{2}$; upon hitting the boundary $\{-25, 25\}$, it reflects. The random walk is initialized randomly over $\{-10, ..., 10\}$. The i^{th} character image, where $i \in \{1, 2, \dots, n_c\}$, will be placed vertically by

centering it according to the vertical center of its bounding box, and then shifting it up or down according to the value $v_{(i-1)\times 6}$ of the random walk. This produces $(n = 6, p = \frac{1}{2})$-binomial shifts before reflection. This is also fixed for all K images.

3. For $1, ..., K$

   (a) For each letter in the string, a random character image is chosen uniformly over $\{f'_{1,i}, \ldots, f'_{18,i}\}$, where i corresponds to the given letter.

   (b) The string image is generated by positioning each character image according to the above horizontal distance and vertical displacement parameters.

The data generation algorithm is motivated by the following: the horizontal distance is varied randomly to introduce crowding between some adjacent characters and make the horizontal positions of characters unpredictable, both of which make segmentation more difficult; the vertical displacement is varied to ensure the vertical location of the word is unpredictable, but a random walk is used to introduce dependence between adjacent characters and aid the reader in following the flow of the word; and the font is chosen randomly for each character to ensure the estimated parameters represent an "average" character, rather than a particular font, so that feature detection or pattern recognition becomes difficult. Still, it must be remembered that the main sources of defense against automated attacks come from the original correlated random field and the pixel by pixel randomness in simulating the word so we do not rely just on character crowding as other methods do, but rather use it as one more layer of protection.

Returning to estimation, we let $s^{(i)}, t^{(i)}$ denote the value of pixels s and t in the i^{th} image (of K generated images) and use the unbiased covariance estimator

$$\beta_{s,t} = \frac{1}{K-1} \sum_{i=1}^{K} (x_{s^{(i)}} - \bar{x}_s)(x_{t^{(i)}} - \bar{x}_t) \text{ for } 0 < \rho(s,t) \leq \ell,$$

where $\bar{x}_s = \frac{1}{K} \sum_{i=1}^{K} x_{s^{(i)}}$ is the empirical mean colour of the pixel at site s. We estimate the marginal probabilities as

$$\pi_s(x_s) = \frac{1}{K} \sum_{i=1}^{K} 1_{x_{s^{(i)}} = x_s}, \text{ where } 1_{x_{s^{(i)}} = x_s} = \begin{cases} 1 & \text{if } x_{s^{(i)}} = x_s \\ 0 & \text{otherwise.} \end{cases}$$

2.C.2 KNW-CAPTCHA Generation

We now present the KNW-CAPTCHA generation details, which consists of generating background noise and then simulating the character string, using modified Gibbs sampling with the parameters obtained in Section 2.C.1, *on top of* the background noise.

Introducing background noise is a common technique when generating CAPTCHAs since it introduces *red herring* character shapes that must be removed or ignored by a computer program. Our view is that the best red herrings are actual character pieces. Background noise also makes segmentation more difficult since, for example, vertical projection will not detect gaps between adjacent characters bridged by appropriate background noise, and connected components will view two adjacent characters as one if they are connected by background noise (see e.g. Figures 2.1a, 2.6c).

We generate background noise via the ScatterType algorithm in [15]. While the original intent of the ScatterType algorithm was to produce CAPTCHAs that were human-readable but difficult to crack, our goal is the reverse: produce ScatterType CAPTCHAs that are clearly unreadable to humans yet "readable" to computers, i.e., the character shapes produced will serve as effective red herrings. By being obviously human-unreadable, the background noise will be visually distinct from the actual character string, serving as a form of stenography. Still, the character pieces are often erroneously detected by computer programs as part of the actual character string. The unreadable background noise is generated using the following algorithm.

1. Choose a five-letter character string uniformly, with replacement, over the English alphabet.

2. Apply the ScatterType algorithm using a fixed font and the following parameters:

Cutting Fraction	0.50	Expansion Fraction	0.60
Horizontal Scatter Mean	0.00	Vertical Scatter Mean	0.00
Scatter Standard Error	0.05	Character Separation	0.20

For our purposes, it is sufficient to say that this algorithm cuts each character into large chunks (roughly quadrants), scatters each chunk, and separates each adjacent character by roughly the width of a character. The reader is referred to [15] for a description of the ScatterType algorithm.

Finally, we are ready to generate the KNW-CAPTCHA. We apply Gibbs-like sampling, where we consider background noise as the initial state and use (2.1) to calculate the conditional probabilities of sites in order to re-simulate them. The challenge is to choose and re-simulate the correct sites so that the KNW-CAPTCHA is human-readable but resistant to crack attempts. We consider such a KNW-CAPTCHA to be a "minimally-readable CAPTCHA".

The KNW-CAPTCHA is generated using the following algorithm. See Figure 2.4 for examples.

1. Select a character string, generate a data sample of size $K = 30 \times n_c$, where n_c is the number of characters in the string, and estimate the parameters as described in 2.C.1.

2. Set R, the sites to re-simulate, as follows:

   (a) $S_p = \{s \in \{1, \ldots, M\} \times \{1, \ldots, N\} : \pi_{x_s}(1) > 0\}$, i.e., the sites that have a non-zero probability of being black.

   (b) $S_p^4 = \{s \in \{1, \ldots, M\} \times \{1, \ldots, N\} : s \in \partial_4(p) \text{ for some } p \in S_p\}$, i.e., the sites that are within a distance of $\ell = 4$ from a site that has a non-zero probability of being black.

   (c) To choose R, select $N_G \times n_c$ sites, where $N_G \in \mathbb{N}$ is constant for all characters, randomly and without replacement from S such that the probability of selecting a site from S_p^4 is ten times greater than selecting a site from $S \setminus S_p^4$. [2]

3. Generate the random ScatterType-based noise as described above. Select 400 sites in the same manner as choosing R, and re-simulate each of those

[2]Sites within and near the defining "shape" of a letter are likely to be re-simulated, while others are not, ensuring the background noise is preserved while the character string is sufficiently human-readable. We consider this a Gibbs-like sampler since the goal is not to reach the joint distribution of the KNW-CAPTCHA but to effectively blend the encoded word with the background noise.

sites using only the marginal probabilities (i.e., assuming independence). Take this to be the background noise.

4. Apply the modified Gibbs sampling:

   (a) Take the initial state to be the background noise.

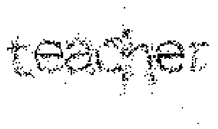
   (b) Re-sample each site in R according to Section 2.B.[3]

This process generates a matrix of black and white pixels saved as a PNM file; in practice, we must use an image format supported by modern web browsers as the CAPTCHAs will typically be deployed on websites. We use ImageMagick [18] to convert the PNM file to a 72 DPI JPEG file, which is used in the following OCR and human readability experiments.

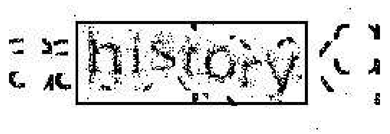
2.D Results

In the following, we describe how we measure the properties of the KNW-CAPTCHA and provide results. In Section 2.D.1, we attack a weak variant of the KNW-CAPTCHA with computer programs to establish a lower-bound to the KNW-CAPTCHAs' attack resistance; in Sections 2.D.2 and 2.D.4, we measure the human readability of the hardened KNW-CAPTCHA; finally, in Section 2.D.3, we measure the attack resistance together with the human readability of the hardened KNW-CAPTCHAs.

[3]Depending on the parameters estimated and the background noise used, we may encounter conditional probabilities outside the bounds of $[0, 1]$. In this paper, we are more concerned with the practical outcome of the algorithm over perfect mathematical sensibility; for this reason, if a probability is encountered outside these bounds, we instead use the marginal probability as a fallback. Please see [17] for a detailed exploration of the constraints on the parameters.



(a) A KNW-CAPTCHA$_{\mathrm{E}}$.



(b) A KNW-CAPTCHA$_{\mathrm{H}}$, with outline.

Figure 2.4: KNW-CAPTCHA examples

We use KNW-CAPTCHA$_{\mathrm{E}}$ to refer to the easy KNW-CAPTCHA variant (Figure 2.4a). This variant is generated with no background noise and no vertical displacement of individual characters, and is designed to be as easy as possible to attack while maintaining the fundamental properties of the KNW-CAPTCHA. The hardened variant, KNW-CAPTCHA$_{\mathrm{H}}$, generated with both background noise and random vertical displacement, is deployed in practice. See Figure 2.4b.

2.D.1 KNW-CAPTCHA$_{\mathrm{E}}$ Experiments

Recall that in a (α, β, η)-CAPTCHA, we want η to be low. We now show our η is low by establishing that modern OCR programs are unable to recognize the encoded words. In particular, we use KNW-CAPTCHA$_{\mathrm{E}}$ and design each experiment to give the attacks the best chance of success. The resistance to attacks in these easy cases is a lower bound for the hardened KNW-CAPTCHA used in practice. However, our results below show even the KNW-CAPTCHA$_{\mathrm{E}}$ is basically unbreakable with contemporary OCR programs.

In order to understand the effect of N_G, the proportion of sites to resimulate in our modified Gibbs sampling, we perform the following experiments over a number of values of N_G and expect N_G to be related to how readable by both computer programs and humans the resulting image is.

We use two OCR programs: Tesseract and ABBYY FineReader. Tesseract is available at `http://code.google.com/p/tesseract-ocr/` (retrieved 2010-09-14). To our knowledge, Tesseract is the best available open-source OCR. An overview of the implementation of Tesseract is given in Smith [19]. ABBYY FineReader is a proprietary OCR program used in, for example, [13].

We proceed in the spirit of giving the OCR programs a "fighting chance" by using KNW-CAPTCHA$_{\mathrm{E}}$. In essence, we make the KNW-CAPTCHA as easy as possible to recognize (while maintaining its fundamental construction). This tactic will provide the most evidence that η is low, i.e., that the KNW-CAPTCHA is difficult to crack. Word accuracy is calculated based on the number of words recognized, and all word comparisons are done ignoring case.

For a particular word, the experiment is as follows.

1. Generate a KNW-CAPTCHA$_{\mathrm{E}}$ for the word word$_K$ with no background noise and no vertical displacement.

2. Run the OCR program to obtain word$_O$.

3. Compare word$_K$ and word$_O$.

We vary N_G and obtain the results over n_T trials under each given value of N_G. Under a given N_G, we model each attempt to recognize the word as i.i.d. (p_w)-Bernoulli random variables, where p_w is the probability of recognizing the word. We use the maximum likelihood estimator $\hat{p}_w$ and provide the 95% confidence interval. We perform the experiment for n_T words, selected without replacement randomly from our dictionary. In order to validate the human readability of the KNW-CAPTCHA$_{\mathrm{E}}$, we also collect human results via Amazon Mechanical Turk (AMT) [20] (see Section 2.D.2 for details). Results are summarized in Table 2.1, and an example of a KNW-CAPTCHA$_{\mathrm{E}}$ is provided in Figure 2.4a. Based on these results, it appears that both OCR programs have great difficulty recognizing the KNW-CAPTCHA$_{\mathrm{E}}$s. In fact, the computer performance on the unhardened KNW-CAPTCHA$_{\mathrm{E}}$ with $N_G = 200$ is similar to the results on the Google CAPTCHA (Table 2.3), which was the most difficult for OCR to recognize of Google, YAHOO, and eBay. In addition, human performance on the KNW-CAPTCHA$_{\mathrm{E}}$ is very high; taken together, this experiment strongly indicates that η is low while β is high, as desired. As expected, both OCR and human performance generally increase as N_G increases, which indicates that N_G will serve an important role in balancing readability and security.

N_G	n_T	ABBYY	Tesseract	Human
200	1000	0.012 ± 0.007	0.000 ± 0.000	0.990 ± 0.006
400	1000	0.015 ± 0.008	0.000 ± 0.000	0.991 ± 0.006
600	1000	0.007 ± 0.005	0.002 ± 0.003	0.991 ± 0.006
800	1000	0.032 ± 0.011	0.015 ± 0.008	0.996 ± 0.004
1000	1000	0.034 ± 0.011	0.020 ± 0.009	0.993 ± 0.005

Table 2.1: 95% Confidence Interval of Computer and Human $\hat{p}_w$

2.D.2 KNW-CAPTCHA$_{\mathrm{H}}$ Experiments

In the (α, β, η)-CAPTCHA context, our β is high. The KNW-CAPTCHA should be applied to literate English-reading adults with normal eyesight. (In practice, alternative CAPTCHAs, such as an audio CAPTCHA, should be provided others.) Our task is to estimate β and the time to complete the challenge empirically.

The following experiments use a set of 300 KNW-CAPTCHA$_\mathrm{H}$ images generated with $N_G = 800$ based on the results of the previous sections along with visual inspection in order to balance attack resistance with readability.

Online Readability Study

To collect these results, we set up the website `http://www.knwcaptcha.org`. Volunteers participating in this online study were anonymous. No incentive was provided. The procedure was as follows.

1. The visitor is presented with information on how the experiment is conducted and how the data will be used. If the user does not accept, the experiment is terminated.

2. In order to familiarize the visitor with the process, he or she is presented with an example of a KNW-CAPTCHA$_\mathrm{H}$ along with the correct response. The example shows a KNW-CAPTCHA$_\mathrm{H}$ with the encoded word outlined, and is designed to show the visitor how to recognize the encoded word in noise. See Figure 2.4b.

3. The visitor is shown a set of 25 KNW-CAPTCHA$_\mathrm{H}$s. A visitor is never shown the same word more than once. Beside each KNW-CAPTCHA$_\mathrm{H}$, the visitor enters a response, and submits the entire data set upon completion.

A human's response to a KNW-CAPTCHA$_\mathrm{H}$ is marked as correct if it matches the encoded word, ignoring case, and incorrect otherwise. In the analysis, we model the trials as i.i.d. (β)-Bernoulli random variables. The experiment yields n_T responses from humans $y_1, \ldots, y_{n_T}$, where

$$y_i = 1_{i^{\text{th}} \text{ response was correct}}.$$

As before, we use the maximum likelihood estimator $\hat{\beta}$ to estimate β. The time to solve each challenge is calculated using the time elapsed from when the user is first presented with the CAPTCHAs to the submission of the responses. The results are summarized in Table 2.2. Humans succeeded at solving a high proportion of KNW-CAPTCHA$_{\text{H}}$s quickly, helping to establish that β is high and our CAPTCHAs are not onerous.

Amazon Mechanical Turk

In addition to collecting responses from volunteers at `knwcaptcha.org`, we used Amazon Mechanical Turk (AMT) [20]. AMT is an online service which allows requesters to submit tasks which will be completed by a pool of workers. The use of AMT for collecting human feedback in research has been established in several works. In Kittur *et al.* [21], the authors found that high quality responses are achievable when using an appropriate experimental design; for example, in order to be resistant to workers gaming the task, it is important that the task be as much effort to complete incorrectly as correctly. In Sorokin *et al.* [22], the authors successfully use AMT for the purpose of image annotation. In Bursztein *et al.* [1], a number of popular CAPTCHA schemes are evaluated in terms of human readability based on the amount of agreement between three workers on a CAPTCHA image containing an unknown word. (For further discussion of [1], please see Section 2.D.4.) Our task of evaluating the responses to a known CAPTCHA is relatively straightforward and an appropriate fit for AMT.

Our AMT task design is similar to that of `knwcaptcha.org`. Each task submitted to AMT consisted of a KNW-CAPTCHA$_{\mathrm{H}}$ image and a response field. A batch of tasks is preceded by brief instructions and an example, as on `knwcaptcha.org`. No qualification pre-tests are administered, nor are workers penalized (via, for example, lack of payment) for wrong answers. AMT provides more diverse, international respondents than could be obtained by recruiting local volunteers as in Section 2.D.2. While no demographic information is collected, a comprehensive survey of the AMT worker population conducted by Ross *et al.* [23] found a large population of international, young, educated workers. Furthermore, [1] examines the effect of demographics on CAPTCHA solving ability; of particular interest to us is that native English speakers are able to solve English or pseudo-English CAPTCHAs far faster, which indicates that the KNW-CAPTCHA$_{\mathrm{H}}$ is biased against non-native English speakers.

The responses collected are summarized in Table 2.2. As before, we model the trials as i.i.d. (β)-Bernoulli random variables, and solving time is calculated as in Section 2.D.2. While there is a drop in accuracy when compared with the results in Section 2.D.2, this is likely explained by the different demographics of the respondents, particularly native language, as well as the lack of incentives for correct responses. Unsurprisingly, the AMT workers, who are incentivized to complete tasks quickly, solve the CAPTCHAs faster than their volunteer counterparts.

2.D.3 OCR Attacks on Hardened KNW-CAPTCHAs

Next, we confirm that the KNW-CAPTCHA$_\mathrm{H}$ is an effective separator of humans and computer programs by providing an "apples to apples" comparison of OCR performance against human performance. To obtain these results, we ran Tesseract and ABBYY FineReader on the KNW-CAPTCHA$_\mathrm{H}$s for which we have human responses and determined accuracy as before. The results are in Table 2.2.

As is clear from Table 2.2, neither OCR program is able to recognize any of the KNW-CAPTCHA$_\mathrm{H}$s, while humans perform remarkably well on them. In fact, the OCR programs seldom recognized any of the characters present in the word. These results, taken together with the results in Section 2.D.1, provide strong evidence that the KNW-CAPTCHA$_\mathrm{H}$ defends automated attacks well while also remaining quickly and easily solvable by humans. In particular, we see that the human time to solve the KNW-CAPTCHA$_\mathrm{H}$ is low so our CAPTCHAs are not onerous.

	N_G	n_T	95% Confidence Interval	Time to Solve
knwcaptcha.org	800	300	0.960 ± 0.022	6.41s
AMT	800	3319	0.910 ± 0.010	4.98s
Tesseract	800	300	0.000 ± 0.000	N/A
ABBYY	800	300	0.000 ± 0.000	N/A

Table 2.2: KNW-CAPTCHA$_\mathrm{H}$ Human and Computer Performance

2.D.4 Comparison

We now compare the KNW-CAPTCHA$_\mathrm{H}$ to other popular CAPTCHAs by replicating the procedure used in the excellent CAPTCHA readability study in Bursztein *et al.* [1].

In [1], responses are collected from three distinct AMT workers for each CAPTCHA image. Since the correct answer for each CAPTCHA is unknown, they instead compute "optimistic solving accuracy": for a particular CAPTCHA image, if all three responses agree then all three responses assumed to be correct; if two agree, then two responses are assumed to be correct; otherwise, one response is assumed to be correct. In addition, we collect responses from ABBYY and Tesseract as before; in this case, an OCR program is considered correct if it matches any of the three human responses. See Table 2.3 for optimistic computer and human solving accuracy, where n_C is the number of CAPTCHAs used, and n_H is the number of human responses collected.

	n_C	ABBYY	Tesseract	n_H	$\hat{\beta}$
KNW-CAPTCHA$_{\mathrm{H}}$	300	0.00 ± 0.00	0.00 ± 0.00	900	0.94 ± 0.02
Google	300	0.00 ± 0.00	0.01 ± 0.01	900	0.81 ± 0.03
YAHOO	300	0.01 ± 0.01	0.02 ± 0.01	900	0.93 ± 0.02
eBay	300	0.05 ± 0.02	0.29 ± 0.05	900	0.97 ± 0.01

Table 2.3: 95% Confidence Interval of Optimistic Computer and Human Performance

The KNW-CAPTCHA$_{\mathrm{H}}$ compares favourably with the Google, YAHOO, and eBay CAPTCHAs: it is the only CAPTCHA that was unrecognized by either OCR program, and only the eBay CAPTCHA was more human readable (though the eBay CAPTCHA also appears trivially broken). The Google CAPTCHA was seldomly recognized by OCR, but at significant human readability cost.

The optimistic human solving accuracy on the KNW-CAPTCHA$_{\mathrm{H}}$ also compares favourably to many popular CAPTCHA schemes used in [1], including reCAPTCHA (0.75), Google (0.86), and Yahoo (0.88). A few CAPTCHA schemes achieved higher accuracy, like the Authorize CAPTCHA (0.98). How-

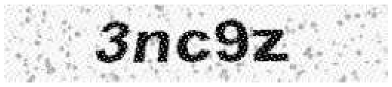


Figure 2.5: Authorize CAPTCHA Example [1]

ever, this study did not assess attack resistance; the Authorize CAPTCHA example in Figure 2.5 is straightforward to segment, for example. The reader is referred to [1] for details.

2.E Security Discussion

Removal of noise is typically the first step of a CAPTCHA attack, and is often straightforward due to the noise's distinctness from the character images, as in the PayPal CAPTCHA [11] and Gimpy-r [12]. Dictionary knowledge facilitates specific pattern discovery in many text-based CAPTCHAs, as in the bigram-based attack against Gimpy [2]. Font knowledge can be used to determine the most likely character for a particular character image, as in [12]. Finally, segmentation of word images was often a critical step in order to individually attack and recognize characters, as in [11] and [12]. We now examine how the KNW-CAPTCHA$_{\mathrm{H}}$ resists like-minded attacks.

The KNW-CAPTCHA$_{\mathrm{H}}$ uses ScatterType background noise comprised of character fragments difficult to differentiate from the encoded CAPTCHA characters, in contrast to the Microsoft CAPTCHA [9], which relied on random arcs. However, the background still retains distinct qualities; in particular, it does not appear as "noisy" as the encoded characters; which works in our favor for untargeted attacks. If an attack were to target this feature, the amount of degradation done to the initial state could be varied (see Section 2.E.1 for more details).

The KNW-CAPTCHA$_{\mathrm{H}}$ also uses a fixed English word dictionary for human readability. However, this does enable attackers to use dictionary knowledge to improve attack effectiveness. If this proves a weakness, there are three straightforward alternatives: increase the dictionary size (reCAPTCHA uses 100,000 words [24]); use pseudo-words as in [15]; or use random character strings.

It will prove very difficult for attackers to leverage font knowledge against the KNW-CAPTCHA$_{\mathrm{H}}$. The KNW-CAPTCHA algorithm learns the parameters to simulate a character image from many fonts; no one particular font is used, and the set of fonts used can be changed easily. Instead, each character image in the KNW-CAPTCHA is randomly simulated, leading to a partial, noisy image such that no two realizations are the same nor do they match any of the fonts. This contrasts particular font distortions, as in Gimpy-r, which yielded to distortion estimation techniques [12], or to particular font noise obscurations, as in the Pessimal Print CAPTCHA [13].

Segmentation resistance is critical for CAPTCHA design since a trained computer program can outperform humans at recognizing distorted, cluttered single character images [25]. Segmentation continues to be error-prone for OCR, but several CAPTCHAs have been broken via segmentation attacks (see Section 2.A). As several modern CAPTCHAs, the KNW-CAPTCHA$_{\mathrm{H}}$ uses character crowding (in addition to random images). The KNW-CAPTCHA$_{\mathrm{H}}$ crowds using adjustable random spacing, as in [15], which typically leads to some adjacent character images overlapping.

To illustrate, we implement and execute a vertical projection segmentation attack on the easier KNW-CAPTCHA$_{\mathrm{E}}$. The vertical projection attack, at its simplest, calculates the total number of "on", or black, pixels in each column in an image. The image is then segmented at columns where there are few or no black pixels. This method is very fast since only one pass of the image is required, making it a useful tool for attempting to crack large numbers of CAPTCHAs. We implement a more sophisticated variant of the vertical projection attack to identify segmentation candidates as described by Tsujimoto *et al.* [26], which is designed to segment touching characters as in KNW-CAPTCHAs. (Casey *et al.* [8] provides an excellent overview of segmentation methods.)

Tsujimoto *et al.* [26] define their algorithm for finding segmentation candidates as follows.

1. For each adjacent pair of columns, perform an AND operation and determine the number of black pixels in the resulting column (i.e., the number of pixels that were black in both columns); this number is called the *break cost*.

2. Smooth the break costs obtained in the previous step.

3. Identify break candidates as local minima in the smoothed break costs.

Herein, we smooth using a moving average.

We use the following experimental procedure for measuring the segmentation performance.

1. Generate an easy KNW-CAPTCHA for a character string consisting of two random (i.e., chosen uniformly over the English alphabet), lower case letters, with no background noise (i.e., the modified Gibbs sampling is initialized with a white image).

2. Find the global smoothed break point minimum within the boundaries (i.e., in the horizontal region between the first and last black pixels) of the generated KNW-CAPTCHA using the above sophisticated vertical projection segmentation attack. In the case of a tie, select the point randomly amongst the global minima. Use this as the point of segmentation.

3. If the bounding boxes of the two characters overlap, and the segmentation point is within two pixels of the middle of the overlapping region, then consider the segmentation correct. If the bounding boxes do not overlap, then consider the segmentation point correct if it lies anywhere in the region between the bounding boxes.

The determination of whether the segmentation point is correct is slightly modified from the work done by Hoffman *et al.* [27], which sought to isolate measures of segmentation performance from recognition engines. Since we are attempting to evaluate only segmentation resistance at this point, rather than recognition resistance, ours was an appropriate technique to adopt.

We estimate p_s, the probability of successful segmentation, using the maximum likelihood estimator $\hat{p}_s$. The results are summarized in Table 2.4. The segmentation performance is low despite targeting character crowding and presenting simplified two character images without scatter noise. Furthermore, the segmentation performance does not vary significantly with N_G indicating the vertical projection algorithm has difficulty with the basic construction of the KNW-CAPTCHA.

Collectively, the security mechanisms in the KNW-CAPTCHA$_\mathrm{H}$ will prove difficult to circumvent. However, should it be successfully attacked, other

N_G	n_T	95% Confidence Interval of $\hat{p}_s$
600	1000	0.094 ± 0.018
800	1000	0.092 ± 0.018
1000	1000	0.088 ± 0.018

Table 2.4: Segmentation Performance

variants may take its place.

2.E.1 Variants

The mechanism described in Section 2.C is more general than the particular example we study in this work: it can easily be extended to counter new attacks. For example, if an attacker is able to remove the background noise, one can use striped correlated noise (see Figure 2.6c); if a dictionary-based attack succeeds, one can use pseudo-words; if a segmentation attack succeeds, one can increase character crowding or decrease N_G (see Figure 2.6a). In fact, one could deploy several variants simultaneously, effectively reducing the reward for successfully attacking any particular variant. We now provide a high-level view of potentially useful variants; furthermore, we will discuss how a variant can be selected by a user of the KNW-CAPTCHA.

The algorithm given in Section 2.C consists of the following steps.

1. Sample generation: many random instances of a character string image are generated.

2. Parameter estimation: simulation parameters are estimated from image samples.

3. Initial state: an initial state for the KNW-CAPTCHA is generated.

4. Simulation: re-simulate random pixels of the KNW-CAPTCHA until word appears.

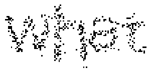
This is a template method pattern [28], meaning that we have given a high-level description of the algorithm while allowing variants to define the details of how each step is accomplished.

Within, we studied two variants, the KNW-CAPTCHA$_{\mathrm{E}}$ and the KNW-CAPTCHA$_{\mathrm{H}}$. The KNW-CAPTCHA$_{\mathrm{E}}$ was deliberately designed to be vulnerable to attack by eschewing random vertical displacement in the sample generation step, and background noise. In contrast, the KNW-CAPTCHA$_{\mathrm{H}}$ does use random vertical displacement and the initial state is generated using a ScatterType CAPTCHA. These two relatively simple differences produce significantly different CAPTCHAs, yet the overall algorithm remains the same.

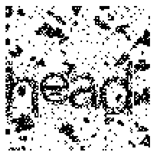
Now we introduce several variants to illustrate the flexibility of the KNW-CAPTCHA algorithm.

Low N_G We use a less cluttered initial state and alter the simulation step by using a lower N_G, $N_G = 100$ say. The intent is quite different from the KNW-CAPTCHA$_{\mathrm{H}}$: instead of relying on background noise to defend against attack, we are relying on using only partially-formed character shapes to ensure segmentation is difficult. See Figure 2.6a.

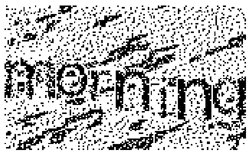
Clustered Correlated Noise Instead of the ScatterType character fragments, we generate the initial state using the simulation algorithm detailed in Section 2.B, with pair-wise covariances set to the Euclidean distance from the pixel being simulated with $\ell = 2$. The effect is an initial state with clustered random shapes. See Figure 2.6b.



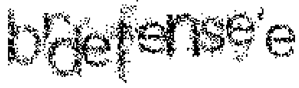
(a) Low N_G



(b) Clustered Correlated Noise



(c) Striped Correlated Noise



(d) Simulated Characters

Figure 2.6: Variants of the KNW-CAPTCHA

Striped Correlated Noise We generate the initial state using one pass of the simulation algorithm with $\ell = 2$. Let the pixel p being simulated have the coordinates (x, y), and let pixel p_i have the coordinates (x_i, y_i). For each pixel p_i in the neighbourhood of p, set the pair-wise covariance to 0 if $x < x_i$ and $y < y_i$, or if $x > x_i$ and $y > y_i$; otherwise set the pair-wise covariance according to the Euclidean distance between p and p_i. This generates striped correlated noise. See Figure 2.6c.

Simulated Characters generate the initial state using the KNW-CAPTCHA$_{\mathrm{E}}$ algorithm with a lower N_G and a random character string. This produces a background noise that is distinct to humans but difficult to eliminate automatically due to the similarity in form to the CAPTCHA word. A random character string is used instead of a word to prevent confusion between the background noise and the CAPTCHA word. See Figure 2.6d.

Variant Selection

While it is clear that it is easy to generate varied CAPTCHAs using the methods laid out in this work by modifying parameters or the steps in the CAPTCHA algorithm, we have not yet discussed how these variants can be compared and selected. In the following, we will present an idea of how to accomplish this automatically.

Any comparison should naturally take into account both attack resistance and human readability. However, different users of CAPTCHAs will place different on each quality and a CAPTCHA should be able to balance the two qualities. More precisely, let $f(\theta) = w \times a(\theta) - (1-w) \times h(\theta)$, where θ is the set of parameters determining the variant of KNW-CAPTCHA generated, $a(\theta)$ is the probability of an attack succeeding on an instance of the variant, $h(\theta)$ is the probability of a human being able to read an instance of the variant, and $w \in [0, 1]$ is a weight balancing the two qualities. Then $f(\cdot)$ is a cost function, and the goal of a CAPTCHA should be to minimize it. The role of w is to allow the user of a CAPTCHA to balance attack resistance and readability. $a(\theta)$ and $h(\theta)$ are unknowable and can only be estimated. One method of estimating $a(\theta)$ would be to attempt to attack the many instances of the CAPTCHA with several OCR engines and consider it a success if any of them succeed (similar to how the reCAPTCHA project determines if a word image should be used as a CAPTCHA challenge [24]), i.e., $\hat{a}(\theta) = \frac{1}{n}\sum_{i=1}^{n} y_i$ where $y_i = 1$ if any of the OCR engines recognize the word, and $y_i = 0$ otherwise. $h(\theta)$ could be estimated in a similar fashion, using human readability experiments. Then, selecting the appropriate CAPTCHA variant becomes a matter of minimizing the cost function over the evaluated variants, i.e., $\theta* = \min_{\theta \in \Theta} \hat{f}(\theta)$.

2.F Conclusion and Future Work

We developed and implemented a new method of generating random CAPTCHAs, called KNW-CAPTCHAs, using random field simulation that outperforms popular CAPTCHAs in use today. First, we estimated the marginal probabilities of sites and site-to-site covariances of the KNW-CAPTCHA based on randomly generated samples; second, we used an efficient algorithm to simulate a new KNW-CAPTCHA based on these parameters in a Gibbs-like manner.

Furthermore, we established that the KNW-CAPTCHA is an effective separator of computer programs and humans. We provided evidence that the KNW-CAPTCHA is difficult for computer programs to crack through an analysis of its resistance to segmentation attacks and OCR attacks. We also established that the KNW-CAPTCHA is very readable to humans.

Finally, we discussed targeted attacks against the KNW-CAPTCHA and several implementation variants, as well as how to select a variant automatically based on empirical results.

There are several methods of further hardening the KNW-CAPTCHA, which we explored in part in Section 2.E.1. Characteristics of the generated CAPTCHA can be varied within the CAPTCHA. For example: the number of sites to re-simulate per character could increase with each character in a KNW-CAPTCHA; the colors used for the background noise and the CAPTCHA could change from left to right; or the amount of noise could be increased or decreased vertically. The intent of these changes would be to effectively add another dimension to the problem, further confusing an attacker without compromising readability.

Finally, in this work we used only black and white when generating samples; however, this method can be readily extended to generate CAPTCHAs with one or many grey levels. As above, the intent would be to increase the dimensionality of the problem for the attacker without decreasing readability; for example, grey levels could be used to make distinguishing between the background and the letters themselves more difficult, or to make the shapes of the characters themselves less obvious. The main challenge would be to adjust the random sample generation and parameter estimation methods used in this paper in such a way that maintains or improves readability.

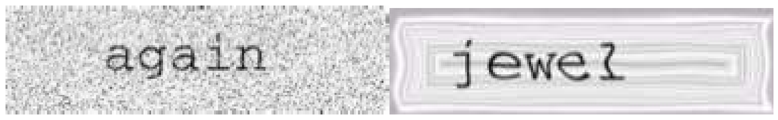
Chapter 3

On Grey Levels in Random CAPTCHA Generation[1]

3.A Introduction

Herein, we extend the KNW-CAPTCHA to grey levels. Where we previously generated KNW-CAPTCHAs with only black and white, we now add a third level that will fall in between. The main goal of this work is to determine if adding a grey level to the KNW-CAPTCHA results in a more effective CAPTCHA, where effectiveness is a measure of both the attack resistance and human readability of the generated CAPTCHA. We hypothesize that adding grey levels to the KNW-CAPTCHAs will increase attack resistance by adding yet another dimension to the problem of OCR (i.e., how grey level should be interpreted), while providing more clues to a human reader about the character form and inter-character separation. The addition of grey levels requires sig-

[1]A version of this chapter has been published. *Proceedings of SPIE Visual Information Processing XX*, vol. 8056, 2011, pp. 80 560U–80 560U–12.



(a) (b)

Figure 3.1: Examples of EZ-Gimpy CAPTCHAs using grey levels [2].

nificant modification to the KNW-CAPTCHA-generation procedure described in [3]. In particular, the sample word image samples used in parameter estimation require grey levels; this is accomplished by overlapping random character pairs, where the presence of black and grey is determined by the regions of overlap. Furthermore, the generation of background noise using ScatterType is also extended to include grey levels. Parameter estimation and the simulation of the KNW-CAPTCHA using Gibbs-like sampling follows in much the same way as in [3].

Use of grey levels or colours in CAPTCHAs are well established in practice; for example, see Figure 3.1. However, we were unable to locate any research on quantifying the impact of the use of multiple colours or grey levels on the effectiveness of the CAPTCHA, which is what we set out to do in this work.

The main goals of this paper are to empirically assess the effectiveness of the KNW-CAPTCHA and determine if black and white or grey-level KNW-CAPTCHAs are the most effective, and to give a sensible approach to selecting KNW-CAPTCHA parameters to balance false positives with false negatives. Furthermore, we aim to provide a detailed analysis of the impact of the parameters in the KNW-CAPTCHA-generation process, including their relation to attack resistance and human readability.

The remainder of this work is laid out as follows. Section 3.B details the changes to the work in [3] required to introduce grey levels to the KNW-CAPT-

CHA; Section 3.C explains our procedure for optimizing the effectiveness of the KNW-CAPTCHA, as measured by a flexible cost function; Section 3.D gives the results of the optimization procedure; a detailed analysis of the results are given in Section 3.E; human readability and attack resistance of the hardened grey-level KNW-CAPTCHA, which would be used in practice, is given in Section 3.F; finally, Section 3.G contains our conclusion and discussion of further work.

3.B Method of Generating the KNW-CAPTCHA

3.B.1 Method of Simulating a Random Field

We provide the required definitions in order to make sense of our simulation formula, equation (3.1) below. In this setting, the random field we want to simulate is a rectangular $M \times N$ image, made up of sites $S = \{(i,j) : 1 \leq i \leq M, 1 \leq j \leq N\}$. We divide the image into an unknown part $H \subset S$ and known part $H^C \circeq S \setminus H$.

We enumerate the sites S column by column: $s_1 = (1,1)$, $s_2 = (2,1)$, $\dots$, $s_{MN} = (M,N)$. Now, we let L be the number of sites in H and enumerate the sites in H column by column, i.e.,

$$h_i = s_{m_i} \ \forall i \in \{1, 2, ..., L\},$$

where

$$m_i = \min\{j > m_{i-1} : s_j \in H\} \text{ and } m_0 = 0.$$

The sites in H will be simulated in the above order.

Next, define the ℓ-neighborhood of site $(i,j) \in S$

$$\partial_\ell((i,j)) = \{(u,v) \in S : 0 < \rho((i,j),(u,v)) \leq \ell\},$$

where $\rho((i,j),(u,v)) = \sqrt{(i-u)^2 + (j-v)^2}$. Then, for each k, we let

$$A_{h_k} \doteq \partial_\ell(h_k) \bigcap \left(\{h_1, h_2, ..., h_{k-1}\} \bigcup H^C\right),$$

i.e., A_{h_k} is the set of sites that are in the neighbourhood of h_k and were already known or have already been simulated.

Having established the prerequisite definitions, the following equation gives us exactly how to compute the conditional probability of a site given its neighbouring sites. Let $\mathbf{X} = \{-1, 0, 1\} = \{\text{white}, \text{grey}, \text{black}\}$ be the state space of each site in S, and $\mathcal{X}_A \doteq \prod_{s \in A} \mathbf{X}$ for $A \subset S$ $\mathcal{X} \doteq \mathcal{X}_S$. We construct the random field $X = (X_s)_{s \in S}$ on the canonical space $\mathcal{X}$ and let X_A denote the projection of X onto $\mathcal{X}_A$.

Assume the numbers on the RHS of equation (3.1) (to follow) are in $[0,1]$. (The conditions for this to be true are given in [17].) Then, there is a not necessarily unique probability measure Π on $\mathcal{X}$ such that

$$\Pi(X_h = c) = \pi_h(c), \; \forall h \in H, \; c \in \mathbf{X},$$

and

$$\text{cov}(X_h, X_t) = \beta_{h,t}, \; \forall \; t \in \partial_l(h),$$

i.e., with correct marginals and covariances, and

$$\Pi(X_{h_i} = x_{h_i} | \, X_{A_{h_i}} = x_{A_{h_i}}) = \\ \pi_{h_i}(x_{h_i}) + \frac{\sum_{t_i \in A_{h_i}} (x_{h_i} - \bar{\mu})\beta_{h_i,t_i}(x_{t_i} - \bar{\mu})}{d^{|A_{h_i}|+1}(\bar{\sigma}^2)^2 \Pi(X_{A_{h_i}} = x_{A_{h_i}} | X_{H^C} = x_{H^C})} \tag{3.1}$$

for each $x_{h_i} \in \mathbf{X}$ and $x_{A_{h_i}} \in \mathcal{X}_{A_{h_i}}$ $(1 \le i \le n)$, where $d = |\mathbf{X}|$, $\bar{\mu} = \frac{1}{d}\sum_{c \in \mathbf{X}} c$, $\bar{\sigma}^2 = \frac{1}{d}\sum_{c \in \mathbf{X}} (c - \bar{\mu})^2$ and $| \cdot |$ is the cardinality of a set.

Simulation of a particular site h_k follows immediately from equation (3.1) as follows.

1. Compute the value of $\Pi(X_{h_k} = c | X_{A_{h_k}} = x_{A_{h_k}})$ for $c \in \mathbf{X}$

2. Generate a $[0, 1]$-uniform random number U to select which value to use for site h_k. In particular, if

   $$\sum_{u=1}^{w-1} \Pi(X_{h_k} = c^u | \, X_{A_{h_k}} = x_{A_{h_k}}) \le U < \sum_{u=1}^{w} \Pi(X_{h_k} = c^u | \, X_{A_{h_k}} = x_{A_{h_k}})$$

   for some $1 \le w \le d$, then we set $X_{h_k} = c^w$, i.e., the realization of X_{h_k} is colour $c^w \in \{-1, 0, 1\}$.

The procedure laid out in Section 3.B.3 requires simulating sites efficiently in a Gibbs-like manner. For the sake of brevity, it is sufficient to say that simulating a site s involves extracting a reduced image consisting of the site s and its neighbourhood. Working with this reduced image, every site is considered to be a neighbour of every other site; this allows for fast computation of the joint probability in the denominator of (3.1) using the multiplication rule. The reader is referred to [3] for the details of this method.

3.B.2 Parameter Estimation

In order to apply (3.1) to generating KNW-CAPTCHAs, we must first estimate the required parameters $\pi_h(c)$ $\forall h \in H$ and $\beta_{h,s}$ $\forall h \in H,\ s \in S$. The procedure is largely the same as in [3], so we will give a brief description and highlight the differences. For a particular KNW-CAPTCHA and word, we estimate the parameters from a sample of K automatically generated word images. Each word image is constructed by assembling a series of character images. In order to ensure consistent placement of characters, we again work with trimmed images of {i,j,l,r,t} and scaled images of other characters in order to ensure all images of a particular character have the same width. In this context, trimming the character images means extra white space around the character is removed so all images of a particular character have a common width. For a given character string, we generate K $M \times N$ images representing that string. A particular image is constructed in the following way.

1. The horizontal distance between adjacent characters is randomly selected from $\{1, 2, 3\}$.

2. The vertical distance between adjacent characters is determined by a reflecting one dimensional random walk.

3. Each character image for the string is randomly selected over the available grey-level images of that character. (The grey-level images are described below.)

4. The character images are concatenated into one image according to the random horizontal and vertical placements determined above.

Let s_k^i denote the k^{th} site in the i^{th} word image in this sample. As in [3], we use the unbiased estimators

$$\beta_{k,t} = \frac{1}{K-1}\sum_{i=1}^{K}(x_{s_k^i} - \bar{x}_{s_k})(x_{s_t^i} - \bar{x}_{s_t}),$$

where $\bar{x}_{s_k} = \frac{1}{K}\sum_{i=1}^{K} x_{s_k^i}$ is the empirical mean, for all $k, t = 1, ..., MN$ and

$$\pi_{s_k}(x_{s_k}) = \frac{1}{K}\sum_{i=1}^{K} 1_{x_{s_k^i} = x_{s_k}},$$

where

$$1_{x_{s_k^i} = x_{s_k}} = \begin{cases} 1 & \text{if } x_{s_k^i} = x_{s_k} \\ 0 & \text{otherwise.} \end{cases}$$

Next, we establish exactly how the grey-level images are constructed from a set of black and white character images. Given two black and white images of the same character, we create a third grey-level image. Let $x_{s_k}^i$ denote the k^{th} pixel of the i^{th} black and white image and let x_{s_k} denote the k^{th} pixel of the grey-level image. Set

$$x_{s_k} = \begin{cases} 1 & \text{if } x_{s_k}^i = 1 \text{ and } x_{s_k}^j = 1 \\ -1 & \text{if } x_{s_k}^i = -1 \text{ and } x_{s_k}^j = -1 \\ 0 & \text{otherwise} \end{cases},$$

i.e., the new grey-level image will be black where the black and white images are both black, white where both are white, and grey otherwise. See Figure 3.2 for an example.

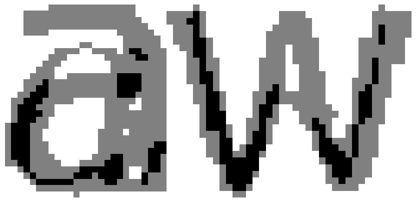

(a) "a" (b) "w"

Figure 3.2: Examples of grey-level character images.

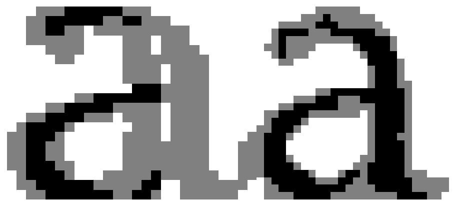

(a) (b)

Figure 3.3: Examples of excluded grey-level character images.

Remark 3.B.1. An interesting feature of this process is that we are now working with character images that do not map easily to any particular font, which should have the effect of making feature detection or pattern matching much more difficult.

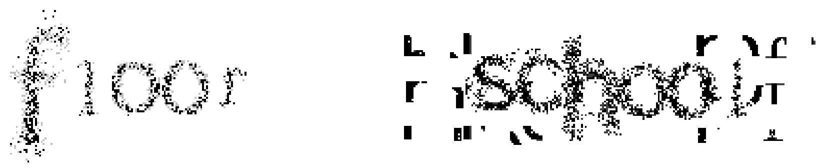
Remark 3.B.2. We automatically exclude certain grey-level images which may be easily recognized by OCR. For a given grey-level image, set all grey pixels to black; if this new image is recognized by the OCR program Tesseract, available at `http://code.google.com/p/tesseract-ocr/`, exclude the grey-level image from the database. Repeat this procedure with all grey pixels set to white. The intent of excluding these images is to help ensure the introduction of grey levels is not trivially bypassed by setting the grey pixels black or white. See Figure 3.3 for examples.

It is clear from Figure 3.2 that the amount of overlap between the two black and white images will certainly have an effect on both readability and attack resistance. In the following, the amount of overlap for a particular grey-level image is calculated as $\frac{\sum_{s_k \in S} 1_{x_{s_k}=1}}{\sum_{s_k \in S} 1_{x_{s_k} \neq -1}}$, i.e., amount of black/amount not white. The effect of overlap will be analyzed in Section 3.E.

3.B.3 Generating a KNW-CAPTCHA

Now that we have estimated the parameters, we describe how to generate a KNW-CAPTCHA for a particular challenge. Again, the procedure is largely the same as in [3], so we give a summary of the approach and highlight the differences. Generating a KNW-CAPTCHA begins with initializing the image with background noise using an implementation of the ScatterType CAPTCHA. Background noise is introduced in order to further complicate attacks on the KNW-CAPTCHA by serving as *red herring* character shapes. From this initial state, we simulate the random field from the parameters estimated in Section 3.B.2. Simulation is accomplished site-by-site for a given number of sites via the Gibbs-like sampling described in 3.B.1. Two passes of the Gibbs-like sampling are used; the first pass is done with a lag of 0, i.e., all sites are considered independent; the second pass is done with a lag of 4. The first pass is used to help ensure that the site-site correlations introduced in the second pass are between sites representing the actual characters instead of between a character site and a site representing the background noise.

As in [3], we use ScatterType to generate the background noise. In essence, for each character in a string, ScatterType cuts the character image randomly and scatters the resulting character pieces randomly; these scattered character images are then concatenated into a single word image [15]. In this work, we now use the grey-level character image database generated in 3.B.2; in fact, we



(a) KNW-CAPTCHA$_E$ (b) KNW-CAPTCHA$_H$

Figure 3.4: Examples of grey-level KNW-CAPTCHAs.

also use grey-level character images that fall into the same target overlap range as our KNW-CAPTCHA. This is intended to ensure that the background noise and simulated character images are not obviously different, and so it is difficult to remove the background noise heuristically.

In the remainder of this paper, we will work with two variants of the KNW-CAPTCHA. The KNW-CAPTCHA$_E$, an easy variant, is generated without background noise or vertical displacement of the characters; the KNW-CAPTCHA$_H$, a hardened variant, is generated with both background noise and vertical displacement. The KNW-CAPTCHA$_E$ is used in cases when we are attempting to measure the relative attack resistance of the generated CAPTCHAs; in our experience, OCR always fails to recognize KNW-CAPTCHA$_H$s, and would provide no information about the relative attack resistance. While the failure rate on the KNW-CAPTCHA$_E$s is still high, it still provides useful results. The KNW-CAPTCHA$_H$ should be used in practice since it is more resistant to attacks See Sections 3.D and 3.E for results using KNW-CAPTCHA$_E$s, and see Section 3.F for readability and attack resistance results of KNW-CAPTCHA$_H$s. See Figure 3.4 for examples of both.

3.C Procedure

We now present our procedure for determining which method of generating KNW-CAPTCHAs is the most effective. In essence, we want to determine if grey-levels provide us with a way of generating more effective KNW-CAPTCHAs. We conjecture that grey will both increase attack resistance and improve human readability. As stated previously, we work with KNW-CAPTCHA$_{\mathrm{E}}$s in order to capture the relative effectiveness of the CAPTCHAs; we will present results for KNW-CAPTCHA$_{\mathrm{H}}$s in Section 3.F.

Before proceeding, we must define "effective". How effectiveness of the CAPTCHA is defined will depend on the goals of the user: are false positives or false negatives worse, or are they equally weighted? (Here, a false positive is claiming a human is a computer program; a false negative is claiming a computer program is a human.) The ideal but unrealistic result is to always deny access to computers, but always allow access to humans. In reality, there will be mistakes; furthermore, there will be a penalty associated with mistakenly allowing a program access (it can abuse whatever resource is being provided) as well as a penalty associated with mistakenly denying access to a human (the site becomes less usable, human visitors can become frustrated and less likely to visit or use the site).

Consider the following diverse examples of CAPTCHA usage. First, a CAPTCHA can be shown after 3 unsuccessful attempts at entering a password, which helps prevent brute force attacks. Second, on an online auction site, preventing bots from accessing the site and automatically bidding may be more important than occasionally denying access to a human since a bot can bid faster and more accurately than humans and could ruin the original intent of the site. Third, a CAPTCHA protecting a low-traffic niche forum might favour allowing humans over denying bots since it would be a low-reward target for a spam bot. We adopt a flexible definition of effective to account for the varying needs.

As previously mentioned, we will work with KNW-CAPTCHA$_{\mathrm{E}}$s, i.e., not hardened (no additional noise, no vertical displacement of characters). For each trial, we generate a KNW-CAPTCHA$_{\mathrm{E}}$ for a random character pair according to a set of randomized parameters. Let N_I be the number of sites re-simulated during the first pass of the Gibbs-like sampling, N_G be the number of sites re-simulated during the second pass of the Gibbs-like sampling, $g \in [0,1]$ be the grey colour used (lower is darker), and $[o_{\min}, o_{\max})$ be the acceptable overlap range for the grey character images used. For each black and white KNW-CAPTCHA$_{\mathrm{E}}$, N_I and N_G will be selected randomly from $\{300, 400, 500\}$. For each grey-level KNW-CAPTCHA$_{\mathrm{E}}$, N_I and N_G will be selected randomly from $\{300, 400, 500\}$, g will be selected randomly from $\{0.25, 0.5, 0.75\}$, and $o_{\min}$ will be selected randomly from $\{0, 0.5\}$ and set $o_{\max} = o_{\min} + 0.5$.

Remark 3.C.1. In the following, we refer to a CAPTCHA's settings as parameters and denote them with θ, while the space of possible parameters is denoted by Θ; our goal is to optimize the parameters, i.e., find the $\theta^* \in \Theta$ which results in the best CAPTCHA. In statistics, the term parameter is often used for quantities which vary over a continuum and will be estimated; in this case, we are using it to refer to the set of inputs which control how the CAPTCHA is constructed and are chosen from a discrete set.

We work with KNW-CAPTCHA$_{\mathrm{E}}$ here since we want to be able to select the best “base” CAPTCHA, then harden it; in addition, it is unlikely that Tesseract would recognize any of the KNW-CAPTCHA$_\mathrm{H}$s, which would make the results useless. We work with random character pairs to generate the minimal, interesting CAPTCHA: single characters do not exercise segmentation-based resistance (which is mainly accomplished through character crowding); words introduce other aspects such as linguistic analysis on the part of Tesseract (see Smith [19] for an overview of Tesseract), as well as being more easily recognizable for humans; however, random character pairs exercise segmentation resistance while also not being easier to recognize due to linguistic analysis. We work with a reduced parameter space based partly on prior work in [3] as well as visual inspection of the generated KNW-CAPTCHAs. The selected parameter ranges were determined to yield generally readable KNW-CAPTCHA’s without being trivial or impossible to crack (i.e., the generated KNW-CAPTCHAs are interesting).

Selecting the most effective KNW-CAPTCHA is now a matter of minimizing a cost function. We use the cost function

$$f(\theta) = w_t \cdot t(\theta) - w_h \cdot h(\theta),$$

where θ are the parameters, $t(\theta)$ is the probability of a KNW-CAPTCHA generated by parameters θ being recognized by the Tesseract OCR program, and $h(\theta)$ is the probability of that KNW-CAPTCHA being recognized by a human, and $w_t, w_h \in [0, 1]$ are weights to adjust the relative importance of false positives and false negatives. For example, setting $w_t = 1, w_h = 0.5$ would make denying access to a computer program twice as important as successfully allowing access to a human. Our goal is to find the θ that minimizes the cost function, i.e., we would like $t(\theta)$, the probability of the KNW-CAPTCHA being cracked, to be low, and we would like $h(\theta)$ to be high.

Of course, we don't know the true functions $t(\theta)$ and $h(\theta)$, so we work with the estimates of the functions $\hat{t}(\theta)$ and $\hat{h}(\theta)$, respectively. First, we estimate $t(\theta)$. For a particular set of parameters θ, model the experimental trials as independent and identically distributed (i.i.d.) $t(\theta)$-Bernoulli random variables. Let

$$y_i = \begin{cases} 1 & \text{if Tesseract's } i^{\text{th}} \text{ response was correct} \\ 0 & \text{otherwise} \end{cases},$$

where the response correctness is determined using a case-insensitive comparison. For a particular set of parameters θ, the experiment yields n_T results, $\{y_1, y_2, \ldots, y_{n_T}\}$. We use parameter-by-parameter maximum likelihood estimator $\hat{t}(\theta)$ for $t(\theta)$, i.e.,

$$\hat{t}(\theta) = \frac{1}{n_T} \sum_{i=1}^{n_T} y_i.$$

We determine $\hat{h}(\theta)$ in exactly the same manner.

Remark 3.C.2. We use the MLE for each unique set of parameters to avoid making any assumptions about the shape of $f(\theta)$ during the optimization process.

Now we optimize the estimated cost function

$$\hat{f}(\theta) = w_t \cdot \hat{t}(\theta) - w_h \cdot \hat{h}(\theta),$$

i.e., we want to find θ^*, the set of parameters which minimizes $\hat{f}(\theta)$. For each unique set of parameters $\theta \in \Theta$, where Θ is the parameter space, determine

$\hat{f}(\theta)$ and choose the parameters which minimize it, i.e.,

$$\theta^* = \text{argmin}_{\theta \in \Theta} \hat{f}(\theta)$$

3.D Results

We now present the results of the procedure laid out in Section 3.C. We give the most effective black and white KNW-CAPTCHA$_\text{E}$ and the most effective grey-level KNW-CAPTCHA$_\text{E}$ for the following cost functions: $w_t = 1, w_h = 1$, where human success and attack failure are considered equally important (Table 3.1); $w_t = 0.5, w_h = 1$, where human success is considered more important than attack failure (Table 3.2); and $w_t = 1, w_h = 0.5$, where attack failure is considered more important than human success (Table 3.3). Similarly, we also include results for $w_t = 0.25, w_h = 1$ (Table 3.4), $w_t = 1, w_h = 0.25$ (Table 3.5), $w_t = 0, w_h = 1$ (Table 3.6), $w_t = 1, w_h = 0$ (Table 3.7); the final two sets of weights are the edge cases that we only care about human readability and only care about attack resistance, respectively. These cost functions all balance the trade-offs of security and human usability differently.

KNW-CAPTCHA$_\text{E}$ Type	Parameters θ^*	$\hat{t}(\theta^*)$	$\hat{h}(\theta^*)$	$\hat{f}(\theta^*)$
Black and White	$N_I = 400, N_G = 400$	0.0082	0.9714	-0.9633
Grey-level	$N_I = 500$, $N_G = 400$, $g = 0.75$, $o_\text{min} = 0.50$, $o_\text{max} = 1.00$	0.0000	0.9860	-0.9860

Table 3.1: Optimization results with $w_t = 1.00, w_h = 1.00$

Section 3.E contains a detailed analysis of results; however, it is immediately clear that the cost function offers a flexible way of selecting a CAPTCHA-

KNW-CAPTCHA$_{\mathrm{E}}$ Type	Parameters θ^*	$\hat{t}(\theta^*)$	$\hat{h}(\theta^*)$	$\hat{f}(\theta^*)$
Black and White	$N_I = 500, N_G = 400$	0.0298	0.9915	-0.9766
Grey-level	$N_I = 500$, $N_G = 400$, $g = 0.75$, $o_{\min} = 0.50$, $o_{\max} = 1.00$	0.0000	0.9860	-0.9860

Table 3.2: Optimization results with $w_t = 0.50, w_h = 1.00$

KNW-CAPTCHA$_{\mathrm{E}}$ Type	Parameters θ^*	$\hat{t}(\theta^*)$	$\hat{h}(\theta^*)$	$\hat{f}(\theta^*)$
Black and White	$N_I = 400, N_G = 400$	0.0082	0.9714	-0.4776
Grey-level	$N_I = 500$, $N_G = 400$, $g = 0.75$, $o_{\min} = 0.50$, $o_{\max} = 1.00$	0.0000	0.9860	-0.4930

Table 3.3: Optimization results with $w_t = 1.00, w_h = 0.50$

generation method, and that the exact choice of w_t, w_h affects the recommended CAPTCHA; in particular, there does not appear to be a universally better KNW-CAPTCHA$_{\mathrm{E}}$ (i.e., one which maximizes attack resistance and human readability simultaneously). Also, in most cases, the recommended CAPTCHA was the grey-level KNW-CAPTCHA$_{\mathrm{E}}$; in fact, the sole exception is the edge case $w_t = 0, w_h = 1$, which would not normally be deployed in practice. Furthermore, the grey-level KNW-CAPTCHA$_{\mathrm{E}}$ was recommended over the black and white KNW-CAPTCHA$_{\mathrm{E}}$ in the case that $w_t = 1, w_t = 1$; indeed, in this case, the grey-level version outperformed the black and white version in both readability and attack resistance.

KNW-CAPTCHA$_\text{E}$ Type	Parameters θ^*	$\hat{t}(\theta^*)$	$\hat{h}(\theta^*)$	$\hat{f}(\theta^*)$
Black and White	$N_I = 500, N_G = 400$	0.0298	0.9915	-0.9840
Grey-level	$N_I = 500$, $N_G = 300$, $g = 0.50$, $o_{\min} = 0.50$, $o_{\max} = 1.00$	0.0058	0.9884	-0.9870

Table 3.4: Optimization results with $w_t = 0.25, w_h = 1.00$

KNW-CAPTCHA$_\text{E}$ Type	Parameters θ^*	$\hat{t}(\theta^*)$	$\hat{h}(\theta^*)$	$\hat{f}(\theta^*)$
Black and White	$N_I = 400, N_G = 300$	0.0040	0.9622	-0.2366
Grey-level	$N_I = 500$, $N_G = 400$, $g = 0.75$, $o_{\min} = 0.50$, $o_{\max} = 1.00$	0.0000	0.9860	-0.2465

Table 3.5: Optimization results with $w_t = 1.00, w_h = 0.25$

3.E Analysis

In the following, we set out to understand how the parameters controlling the generation of the KNW-CAPTCHA$_\text{E}$ impact both the human readability and attack resistance of the resulting CAPTCHA. The following analysis uses logistic regression. Let

$$y_i = \begin{cases} 1 & \text{if } i^{\text{th}} \text{ response was correct} \\ 0 & \text{otherwise} \end{cases},$$

and we model y_i as independent Bernoulli trials and assume that the log odds of the i^{th} response being correct is a linear combination of functions of our i^{th} CAPTCHA's parameter settings, e.g.,

$$\beta_0 + \beta_1 1_{N_I=400} + \beta_2 1_{N_I=500} + \beta_3 1_{N_G=400} + \beta_3 1_{N_G=500}$$

KNW-CAPTCHA$_{\mathrm{E}}$ Type	Parameters θ^*	$\hat{t}(\theta^*)$	$\hat{h}(\theta^*)$	$\hat{f}(\theta^*)$
Black and White	$N_I = 500, N_G = 500$	0.0824	0.9945	-0.9945
Grey-level	$N_I = 500$, $N_G = 300$, $g = 0.50$, $o_{\min} = 0.50$, $o_{\max} = 1.00$	0.0058	0.9884	-0.9884

Table 3.6: Optimization results with $w_t = 0.00, w_h = 1.00$

KNW-CAPTCHA$_{\mathrm{E}}$ Type	Parameters θ^*	$\hat{t}(\theta^*)$	$\hat{h}(\theta^*)$	$\hat{f}(\theta^*)$
Black and White	$N_I = 400, N_G = 300$	0.0040	0.9622	0.0040
Grey-level	$N_I = 300$, $N_G = 400$, $g = 0.50$, $o_{\min} = 0.00$, $o_{\max} = 0.50$	0.0000	0.8361	0.0000

Table 3.7: Optimization results with $w_t = 1.00, w_h = 0.00$

for a black and white KNW-CAPTCHA$_{\mathrm{E}}$ where $\{\beta_j\}_{j=1}^{5}$ are the coefficients which will be estimated via logistic regression. We will interpret the effect of a covariate based on the 95% confidence interval of the odds ratio. Specifically, odds is defined as $\frac{p}{1-p}$, where p is the probability of the response being correct, and odds ratio is defined as

$$\frac{p_1/(1-p_1)}{p_2/(1-p_2)}$$

where p_1 and p_2 are the probabilities of the response being correct under two different sets of CAPTCHA parameter values. For example, if p_1 corresponds to high-N_G CAPTCHAs and p_2 corresponds to low-N_G CAPTCHAs (with other covariates held constant) and the odds ratio is greater than 1, then high N_G would be said to increase the probability of a correct response. Examining the confidence interval of the odds ratio allows us to evaluate both the magnitude and the significance of the effect. We will perform the analysis on the responses from both Tesseract and humans.

Covariate	95% Confidence Interval for Odds Ratio
$1_{N_I=400}$	$[0.56, 2.16]$
$1_{N_I=500}$	$[1.54, 6.38]$
$1_{N_G=400}$	$[0.74, 3.81]$
$1_{N_G=500}$	$[2.86, 10.06]$

Table 3.8: Tesseract Responses on Black and White KNW-CAPTCHA$_\mathrm{E}$

Covariate	95% Confidence Interval for Odds Ratio
$1_{N_I=400}$	$[0.61, 1.67]$
$1_{N_I=500}$	$[1.13, 5.85]$
$1_{N_G=400}$	$[0.84, 2.77]$
$1_{N_G=500}$	$[1.06, 2.58]$

Table 3.9: Human Responses on Black and White KNW-CAPTCHA$_\mathrm{E}$

In general, as is clear from Tables 3.8, 3.9, 3.10, and 3.11, increasing N_I and N_G increases the probability that both Tesseract and humans will recognize a given KNW-CAPTCHA$_\mathrm{E}$. However, the magnitude of the effect appears to be much greater for Tesseract, so these parameters should be kept as low as reasonably possible when correctly rejecting computer agents is important.

Interestingly, increasing g (i.e., using a lighter shade of grey) appears to decrease the probability of Tesseract recognizing the grey-level KNW-CAPTCHA$_\mathrm{E}$, yet does not have a significant effect on human readability, as can be seen in Tables 3.10 and 3.11, respectively. In particular, the lighter grey $g = 0.75$ significantly decreases Tesseract's ability to recognize CAPTCHAs, but does not have a significant affect on humans. This indicates that adjusting the grey level parameter should allow us to design KNW-CAPTCHAs with higher attack resistance without significantly impacting human readability.

As seen in Tables 3.10 and 3.11, it appears that the choice of overlap range significantly impacts human readability (i.e., the more overlap between

Covariate	95% Confidence Interval for Odds Ratio
$1_{N_I=400}$	$[0.48, 32.13]$
$1_{N_I=500}$	$[0.85, 55.73]$
$1_{N_G=400}$	$[1.59, 17.18]$
$1_{N_G=500}$	$[1.93, 12.96]$
$1_{g=0.5}$	$[0.32, 1.91]$
$1_{g=0.75}$	$[0.05, 0.40]$
$1_{[o_{\min}, o_{\max})=[0.5,1.0)}$	N/A

Table 3.10: Tesseract Responses on Grey-level KNW-CAPTCHA$_{\mathrm{E}}$

Covariate	95% Confidence Interval for Odds Ratio
$1_{N_I=400}$	$[0.99, 1.90]$
$1_{N_I=500}$	$[1.46, 3.23]$
$1_{N_G=400}$	$[0.98, 1.91]$
$1_{N_G=500}$	$[0.92, 1.58]$
$1_{g=0.5}$	$[0.81, 2.03]$
$1_{g=0.75}$	$[0.62, 1.44]$
$1_{[o_{\min}, o_{\max})=[0.5,1.0)}$	$[2.69, 4.66]$

Table 3.11: Human Responses on Grey-level KNW-CAPTCHA$_{\mathrm{E}}$

characters, the higher the readability); we were unable to perform this same analysis for Tesseract since the OCR program failed to recognize any of the CAPTCHAs with low overlap. However, this alone is a strong indicator that Tesseract performs poorly on low-overlap CAPTCHAs, and while humans are also negatively impacted, adjusting the acceptable overlap range may also serve to further differentiate between computer programs and humans.

In summary, it appears that grey-level KNW-CAPTCHAs introduce two new parameters, the grey level and the acceptable overlap range, that allow for greater flexibility and power when compared with the black and white KNW-CAPTCHA.

3.F KNW-CAPTCHA$_\mathrm{H}$

Next, we present the attack resistance and readability results of the grey-level KNW-CAPTCHA$_\mathrm{H}$. We generate the KNW-CAPTCHA$_\mathrm{H}$ using the parameters for the most effective grey-level KNW-CAPTCHA$_\mathrm{E}$ on the cost function with $w_t = 1, w_h = 1$. We present these results in order to compare the KNW-CAPTCHA$_\mathrm{H}$ effectiveness with that of the black and white KNW-CAPTCHA$_\mathrm{H}$ presented in [3]. Results are given in Table 3.12.

Parameters	$\hat{t}(\theta)$	$\hat{h}(\theta)$
$N_I = 500, N_G = 400, g = 0.75, o_{\min} = 0.50, o_{\max} = 1.00$	0.0000	0.8639

Table 3.12: Human Readability of KNW-CAPTCHA$_\mathrm{H}$

As expected, Tesseract is unable to recognize any of the generated KNW-CAPTCHA$_\mathrm{H}$s. Surprisingly, however, the human readability is significantly lower than the results on the black and white KNW-CAPTCHA$_\mathrm{H}$ (which obtained a $\hat{h}(\theta)$ of 0.96). It appears that the introduction of background noise is more likely to interfere with the readability when working with the grey-level KNW-CAPTCHAs; this may be due to the increased complexity of the background noise (i.e., the introduction of grey levels into the ScatterType CAPTCHA) when compared with the black and white KNW-CAPTCHA$_\mathrm{H}$s. However, we expect that the readability of the grey-level KNW-CAPTCHA$_\mathrm{H}$ can be increased significantly by adjusting the acceptable overlap range, N_G, and N_I. For now, those wishing to favour human readability over attack resistance may prefer to deploy the KNW-CAPTCHA$_\mathrm{E}$.

3.G Conclusion

- KNW-CAPTCHAs are effective distinguishers of humans over computer programs.
- Adding a grey level to a KNW-CAPTCHA can increase the CAPTCHA's power.
- The actual grey level as well as the amount of grey compared to black are two of the most significant parameters to KNW-CAPTCHA power.
- Adding grey-level background noise makes both human readability and computer recognizability harder over black and white.
- Even grey-level KNW-CAPTCHAs are easy to implement in real time using the given algorithm and they provide effective protection.

Chapter 4

The KNW Random Field

4.A Introduction

We now propose a new class of discrete correlated random fields which incorporate given probability mass functions (pmfs) for all vertices and given pairwise covariances. Proposition 4.B.1 on which our fields are based establishes a method to embed desired covariances and marginal probabilities into a random field while maintaining simulation ease. Indeed, Proposition 4.B.1 is a simple means to construct *some* conditional probabilities consistent with given marginal probabilities and covariances in such a way that sampling the missing portion of a random field sequentially is very feasible. More precisely, when simulating a new vertex, we compute this conditional probability of its state conditioned on the known portion and the previously-simulated vertices, and can construct a random field in one pass based on this algorithm.

4.B Notation and Background

Our goal is to simulate a random field so that desired properties (in our case, marginal probabilities and pair-wise covariances) are maintained. Specifically, we will give a method for computing conditional probabilities so that these properties are maintained. We begin by describing how the problem is constructed in Section 4.B.1; then, we will describe exactly how to compute the probabilities in Section 4.B.2; finally, illustrative examples are given in Section 4.C.

4.B.1 Problem Setup

Suppose we are given a desired probability mass function (pmf) for each random variable and a set of desired pairwise covariances for some set of pairs of the random variables. Our goal is to simulate the random variables so that the desired properties are met.

Definitions

We will be working with undirected and directed graphs in the following. We begin by providing the required definitions and notation.

A graph $G = (V, E)$ is a set of vertices V and edges between those vertices E, where $(u, v) \in E$ if there is an edge between vertices u and v; in this case, u and v are called neighbors. A graph is called connected if there is a path between every pair of vertices. The open neighborhood $N_G(v)$ of vertex $v \in V$ is the set of vertices $u \neq v$ such that there is an edge between u and v, i.e., $(u, v) \in E$. We denote the open neighborhood of v by $N_G(v)$ and the closed neighborhood $N_G(v) \cup \{v\}$ by $N_G[v]$. For a set of vertices B, we define the open neighborhood of B as $N_G(B) = \cup_{v \in B} N_G(v) \setminus B$ and closed neighborhood $N_G[B] = N_G(B) \cup B$. For convenience, we set $N_G(\emptyset) = V$.

An edge in an undirected graph does not have direction associated with it, while an edge in a directed graph (an arc) does. In the following, we will denote undirected graphs with $G = (V, E)$ and directed graphs with $D = (V, A)$.

Setup

We begin by constructing a undirected graph $G = (V, E)$, where V is the set of vertices and E is the set of edges between vertices; there is a vertex for every given random variable and an edge between vertices if there is a given covariance for the corresponding pair of random variables. If the graph G is not connected (a graph is connected if there is a path between every pair of vertices along the given edges), we cannot continue.

Next, we construct a connected directed acyclic graph $D = (V, A)$ from G by traversing G and adding vertices and edges based on the order of traversal. More precisely, we use the following algorithm.

1. Initialise $V_0 = \emptyset$, $A_0 = \emptyset$, and let $N = |V|$ be the number of vertices in G.

2. For $i = 1, \ldots, N$

   (a) Select an arbitrary vertex from $N_G(V_{i-1})$ and denote it as v_i. Note that $N_G(V_{i-1}) \neq \emptyset$ since G is connected.

   (b) Set $V_i = V_{i-1} \cup \{v_i\}$.

   (c) Let $A_{v_i} = \{(v_i, u) : (v_i, u) \in E\}$ be the set of arcs from v_i for every undirected edge involving v_i in G.

   (d) Set $A_i = A_{i-1} \cup A_{v_i}$.

3. Set $A = A_N$. Then $D = (V, A)$ is a connected directed acyclic graph with the same vertices as G and a directed edge for each undirected edge in G.

Remark 4.B.1. The algorithm adds all vertices from G to D since it iterates N times and always adds a vertex from $N_G(V_{i-1})$, i.e., it never adds the same vertex twice. This also ensures that the resulting graph is acyclic.

Remark 4.B.2. Each vertex in D will have a set of parents (u is called a parent of v if there is a directed edge from u to v), which we will denote by $\mathrm{pa}(v)$. The first vertex added to V, v_1, has no parents; all other vertices have at least one parent.

Remark 4.B.3. The order of the vertices $\{v_1, \ldots, v_N\}$ is a topological sort since if $i < j$ and $(v_i, v_j) \in A$, there is an arc from v_i to v_j.

For each vertex $v \in V$, let $\mathbf{X}_v$ be a finite space of states at vertex v. For nonempty $B \subset V$, denote the space of configurations $x_B = (x_v)_{v \in B}$ on B by $\mathcal{X}_B = \prod_{v \in B} \mathbf{X}_v$. We abbreviate $\mathcal{X}_V$ by $\mathcal{X}$, i.e., $\mathcal{X} = \prod_{v \in V} \mathbf{X}_v$. Finally, we denote our given discrete random variables by $X = (X_v)_{v \in V}$ indexed by V. In addition, we denote the desired pmf for each X_v, $v \in V$ by π_v and the desired covariances between X_u and X_v, $(u, v) \in E$ by $\beta_{u,v}$.

Let Π denote a **probability measure** or **distribution** on $\mathcal{X}$. If for every $x \in \mathcal{X}$, $\Pi(x) > 0$, i.e., Π is a strictly positive probability measure on $\mathcal{X}$, then Π is called a **random field**. We also call the random vector $X = (X_v)_{v \in V}$ on the probability space $(\mathcal{X}, \Pi)$ a random field. For a nonempty $B \subset V$, define the projection map from $\mathcal{X}$ onto $\mathcal{X}_B$ as follows:

$$X_B : x \to x_B,$$

where $x \in \mathcal{X}$ and $x_B \in \mathcal{X}_B$.

4.B.2 Method for Computing Conditional Probabilities

Now, we work exclusively with the directed acyclic graph $D = (V, A)$ with N vertices and topological sort $\{v_i\}_{i=1}^N$ and we are ready to assign conditional probabilities.

We place the additional constraint on the distribution Π of X that

$$\Pi(X = x) = \prod_{v \in V} \Pi(X_v = x_v | X_{\mathrm{pa}(v)} = x_{\mathrm{pa}(v)}). \tag{4.1}$$

With this constraint, we are now working with a Bayesian network. Furthermore, we can find the probability of any set of vertices $B \subset V$. Let j be the maximal element of $\{i : 1 \leq i \leq N, v_i \in B\}$, i.e., v_j is the element of B which is last in the topological sort; then, by the multiplication rule and (4.1),

$$\begin{aligned}
\Pi(X_B = x_B) &= \sum_{x_{v_k}: 1 \leq k \leq j, v_k \notin B} P(X_{v_j} = x_{v_j}, X_{v_{j-1}} = x_{v_{j-1}}, \ldots, X_{v_1} = x_{v_1}) \\
&= \sum_{x_{v_k}: 1 \leq k \leq j, v_k \notin B} \prod_{i=1}^{j} P(X_{v_i} = x_{v_i} | X_{v_{i-1}} = x_{v_{i-1}}, \ldots, X_{v_1} = x_{v_1}) \\
&= \sum_{x_{v_k}: 1 \leq k \leq j, v_k \notin B} \prod_{i=1}^{j} P(X_{v_i} = x_{v_i} | X_{\mathrm{pa}(v_i)} = x_{\mathrm{pa}(v_i)}). \qquad (4.2)
\end{aligned}$$

Herein, we simulate random fields with given marginal probabilities for vertices and given covariances between vertices. Our algorithm constructs X with the given marginal probabilities $\{\pi_{v_i}(x_{v_i}) : x_{v_i} \in \mathbf{X}_{v_i}\}_{i=1}^N$ and the given covariances between nearby vertices $\{\beta_{v_i,u} : u \in \text{pa}(v_i)\}$ for $1 \leq i \leq N$. (It is assumed a priori that these marginal conditions hold within V. It is also assumed that $\beta_{v_i,u} = \beta_{u,v_i}$ for $u \in \text{pa}(v_i)$ $(1 \leq i \leq N)$ since $\beta_{v_i,u}$ will denote covariance between X_{v_i} and X_u.) We assign conditional probabilities $\Pi(X_{v_i} = x_{v_i}|X_{\text{pa}(v_i)} = x_{\text{pa}(v_i)})$ such that we maintain the desired covariances and marginal probabilities.

Proposition 4.B.1. Assume that $D = (V, A)$ is a connected directed acyclic graph with N vertices and that $X = (X_v)_{v \in V}$ are discrete random variables indexed by V. Suppose further that $\{v_i\}_{i=1}^N$ is a topological sort of the vertices V, and $\{\tilde{\pi}_v(x_v) : x_v \in \mathbf{X}_v, v \in V\}$ and $\{\hat{\pi}_v(x_v) : x_v \in \mathbf{X}_v, v \in V\}$ are two sets of auxiliary pmfs. Assume that $\{\pi_v(x_v) : x_v \in \mathbf{X}_v, v \in V\}$ are pmfs and $\{\beta_{u,v} : (u,v) \in A \text{ or } (v,u) \in A\}$ are numbers such that the right hand side of (4.3) is in [0,1]. Form the conditional probabilities recusively, starting with $i = 1$, as

$$\begin{aligned}\Pi(X_{v_i} = x_{v_i}|X_{\text{pa}(v_i)} = x_{\text{pa}(v_i)}) &= \pi_{v_i}(x_{v_i}) + \frac{g(v_i)}{\Pi(X_{\text{pa}(v_i)} = x_{\text{pa}(v_i)})} \\ &\quad \times \sum_{u \in \text{pa}(v_i)} \beta_{u,v_i} g(u) h(u, v_i) \qquad (4.3)\end{aligned}$$

for each $x_{v_i} \in \mathbf{X}_{v_i}$ and $x_{\text{pa}(v_i)} \in \mathbf{X}_{\text{pa}(v_i)}$ $(1 \leq i \leq N)$, where $\mu_{\tilde{\pi}_v} = \sum_{x_v \in \mathbf{X}_v} \tilde{\pi}_v(x_v) x_v$, $\sigma^2_{\tilde{\pi}_v} = \sum_{x_v \in \mathbf{X}_v} \tilde{\pi}_v(x_v)(x_v - \mu_{\tilde{\pi}_v})^2$, $g(v) = \frac{\tilde{\pi}_v(x_v)(x_v - \mu_{\tilde{\pi}_v})}{\sigma^2_{\tilde{\pi}_v}}$, $h(u,v) = \prod_{w \in \text{pa}(v) \setminus \{u\}} \hat{\pi}_w(x_w)$, and $\Pi(X_{\text{pa}(v_i)} = x_{\text{pa}(v_i)})$ is computed according to (4.2). Then, the random field

X, defined by

$$\Pi(X = x) = \prod_{i=1}^{N} \Pi(X_{v_i} = x_{v_i} | X_{\mathrm{pa}(v_i)} = x_{\mathrm{pa}(v_i)})$$

has marginal probabilities $\{\pi_v\}$ and covariances $\mathrm{cov}(X_u, X_v) = \beta_{u,v}$ for all $u \in \mathrm{pa}(v)$.

Remark 4.B.4. Proposition 4.B.1 allows us to fully specify the Bayesian network, i.e., we are able to compute $\Pi(X_v = x_v | X_{\mathrm{pa}(v)} = x_{\mathrm{pa}(v)})$ $\forall v \in V, x_v \in \mathbf{X_v}, x_{\mathrm{pa}(v)} \in \mathbf{X}_{\mathrm{pa}(\mathbf{v})}$.

Remark 4.B.5. In Proposition 4.B.1, we assumed that $\pi_v(x_v) > 0 : \forall x_v \in \mathbf{X}_v$ for each $v \in S$. Note that $\mathbf{X}_v$ can be different for each $v \in V$. For given v, if there exists a $x_v \in \mathbf{X}_v$ such that $\pi_v(x_v) = 0$, we may deem it uninteresting and replace $\mathbf{X}_v$ with $\mathbf{X}_v \setminus \{x_v\}$. Therefore the positive probability mass function assumption of π_v is also a convention.

Remark 4.B.6. A random field generated by Proposition 4.B.1 is a correlated random field. Indeed, one value of this proposition is the assertion that there are correlated random fields that match a given collection of marginal probabilities and covariances. We call the random fields generated by Proposition 4.B.1 the **KNW Random Fields** for ease of future reference.

Remark 4.B.7. If $\mathbf{X}_v = \mathbf{X}$ and $d = |\mathbf{X}|$, $\hat{\pi}_v(x_v) = \tilde{\pi}_v(x_v) = \frac{1}{d}, x_v \in \mathbf{X}$ for all $v \in V$ where d is the cardinality of $\mathbf{X}$, then (4.3) takes the following simple form

$$\begin{aligned} \Pi(X_{v_i} = x_{v_i} | X_{\mathrm{pa}(v_i)} = x_{\mathrm{pa}(v_i)}) &= \pi_{v_i}(x_{v_i}) \\ &+ \sum_{u \in \mathrm{pa}(v_i)} \frac{(x_{v_i} - \bar{\mu})\beta_{v_i,u}(x_u - \bar{\mu})}{d^{|\,\mathrm{pa}(v_i)|+1}(\bar{\sigma}^2)^2 \Pi(X_{\mathrm{pa}(v_i)} = x_{\mathrm{pa}(v_i)})} \end{aligned} \tag{4.4}$$

where $\bar{\mu} = \frac{1}{d}\sum_{x_v \in \mathbf{X}} x_v$, $\bar{\sigma}^2 = \frac{1}{d}\sum_{x_v \in \mathbf{X}} (x_v - \bar{\mu})^2$ $(v \in V)$ and $|\operatorname{pa}(v_i)|$ is the cardinality of $\operatorname{pa}(v_i)$. Formula (4.4) is used in [3] for CAPTCHA generation.

Proof. It is clear by (4.3) that for $i = 1$, X_{v_1} has probability distribution $\pi_{v_1}(\cdot)$, since $\operatorname{pa}(v_1) = \emptyset$.

We next prove that for $2 \le i \le N$, X_{v_i} has probability distribution $\pi_{v_i}(\cdot)$, and for any $u \in \operatorname{pa}(v_i)$, $cov(X_{v_i}, X_u) = \beta_{v_i,u}$. To ease notation, we suppress the subscript i. For $x_v \in \mathbf{X}_v$, by (4.3), one has that

$$\begin{aligned}
\Pi(X_v = x_v) &= \sum_{x_{\operatorname{pa}(v)} \in \mathbf{X}_{\operatorname{pa}(v)}} \Pi(X_v = x_v | X_{\operatorname{pa}(v)} = x_{\operatorname{pa}(v)})\Pi(X_{\operatorname{pa}(v)} = x_{\operatorname{pa}(v)}) \\
&= \pi_v(x_v) + g(v) \sum_{u \in \operatorname{pa}(v)} \sum_{x_{\operatorname{pa}(v)} \in \mathbf{X}_{\operatorname{pa}(v)}} h(u,v)\beta_{v,u} g(u) \\
&= \pi_v(x_v) + g(v) \sum_{u \in \operatorname{pa}(v)} \beta_{v,u} \sum_{x_{\operatorname{pa}(v)\setminus\{u\}} \in \mathbf{X}_{\operatorname{pa}(v)\setminus\{u\}}} h(u,v) \sum_{x_u \in \mathbf{X}_u} g(u) \\
&= \pi_v(x_v), \qquad (4.5)
\end{aligned}$$

since for fixed $u \in \operatorname{pa}(v)$,

$$\sum_{x_u \in \mathbf{X}_u} g(u) = \sum_{x_u \in \mathbf{X}_u} \frac{\tilde{\pi}_u(x_u)(x_u - \mu_{\tilde{\pi}_u})}{\sigma^2_{\tilde{\pi}_u}} = \frac{1}{\sigma^2_{\tilde{\pi}_u}}(\mu_{\tilde{\pi}_u} - \mu_{\tilde{\pi}_u}) = 0.$$

Now fix $u \in \operatorname{pa}(v)$, we prove $cov(X_v, X_u) = \beta_{v,u}$. We compute the joint probability mass function of X_v and X_u. For $x_v \in \mathbf{X}_v, x_u \in \mathbf{X}_u$, we have that by (4.3) again

$$
\begin{aligned}
\Pi(X_v = x_v, &X_u = x_u) \\
&= \sum_{x_{\mathrm{pa}(v)\setminus\{u\}} \in \mathbf{X}_{\mathrm{pa}(v)\setminus\{u\}}} \Pi(X_v = x_v | X_u = x_u, X_{\mathrm{pa}(v)\setminus\{u\}} = x_{\mathrm{pa}(v)\setminus\{u\}}) \\
&\quad \times \Pi(X_u = x_u, X_{\mathrm{pa}(v)\setminus\{u\}} = x_{\mathrm{pa}(v)\setminus\{u\}}) \\
&= \pi_v(x_v) \sum_{x_{\mathrm{pa}(v)\setminus\{u\}} \in \mathbf{X}_{\mathrm{pa}(v)\setminus\{u\}}} \Pi(X_u = x_u, X_{\mathrm{pa}(v)\setminus\{u\}} = x_{\mathrm{pa}(v)\setminus\{u\}}) \\
&\quad + g(v) \sum_{x_{\mathrm{pa}(v)\setminus\{u\}} \in \mathbf{X}_{\mathrm{pa}(v)\setminus\{u\}}} \sum_{u \in \mathrm{pa}(v)} h(u,v)\beta_{v,u} g(u)
\end{aligned}
$$

since $\Pi(X_u = x_u, X_{\mathrm{pa}(v)\setminus\{u\}} = x_{\mathrm{pa}(v)\setminus\{u\}}) = \Pi(X_{\mathrm{pa}(v)} = x_{\mathrm{pa}(v)})$ for $u \in \mathrm{pa}(v)$. Therefore,

$$
\begin{aligned}
\Pi(X_v = x_v, &X_u = x_u) \\
&= \pi_v(x_v)\pi_u(x_u) + g(v) \\
&\quad \times \sum_{x_{\mathrm{pa}(v)\setminus\{u\}} \in \mathbf{X}_{\mathrm{pa}(v)\setminus\{u\}}} \left[h(u,v)\beta_{v,u} g(u) + \sum_{w \in \mathrm{pa}(v)\setminus\{u\}} h(u,v)\beta_{v,u} g(u) \right] \\
&= \pi_v(x_v)\pi_u(x_u) + g(v)\beta_{v,u} g(u) \prod_{w \in \mathrm{pa}(v)\setminus\{u\}} \left(\sum_{x_w \in \mathbf{X}_w} \hat{\pi}_w(x_w) \right) \\
&\quad + g(v) \sum_{w \in \mathrm{pa}(v)\setminus\{u\}} \sum_{x_{\mathrm{pa}(v)\setminus\{u\}} \in \mathbf{X}_{\mathrm{pa}(v)\setminus\{u\}}} h(u,v)\beta_{v,u} g(u) \\
&= \pi_v(x_v)\pi_u(x_u) + g(v)\beta_{v,u} g(u) \\
&\quad + g(v) \sum_{w \in \mathrm{pa}(v)\setminus\{u\}} h(w,v)\beta_{v,w} \sum_{x_{\mathrm{pa}(v)\setminus\{u,w\}} \in \mathbf{X}_{\mathrm{pa}(v)\setminus\{u,w\}}} \sum_{x_w \in \mathbf{X}_w} g(w) \\
&= \pi_v(x_v)\pi_u(x_u) + g(v)\beta_{v,u} g(u).
\end{aligned}
$$

Therefore, since

$$\begin{aligned} cov(X_v, X_u) &= cov(X_v - \mu_{\tilde{\pi}_v}, X_u - \mu_{\tilde{\pi}_u}) \\ &= E[(X_v - \mu_{\tilde{\pi}_v})(X_u - \mu_{\tilde{\pi}_u})] - E[X_v - \mu_{\tilde{\pi}_v}][X_u - \mu_{\tilde{\pi}_u}] \end{aligned}$$

and

$$\begin{aligned} E[(X_v - \mu_{\tilde{\pi}_v})(X_u - \mu_{\tilde{\pi}_u})] &= \sum_{x_v \in \mathbf{X}_v} \sum_{x_u \in \mathbf{X}_u} \Big[(x_v - \mu_{\tilde{\pi}_v})(x_u - \mu_{\tilde{\pi}_u}) \\ &\quad \times \Pi(X_v = x_v, X_u = x_u)\Big] \\ &= \sum_{x_v \in \mathbf{X}_v} \sum_{x_u \in \mathbf{X}_u} \Big[(x_v - \mu_{\tilde{\pi}_v})(x_u - \mu_{\tilde{\pi}_u}) \\ &\quad \times [\pi_v(x_v)\pi_u(x_u) + g(v)\beta_{v,u}g(u)]\Big] \\ &= \sum_{x_v \in \mathbf{X}_v} \sum_{x_u \in \mathbf{X}_u} (x_v - \mu_{\tilde{\pi}_v})(x_u - \mu_{\tilde{\pi}_u})\pi_v(x_v)\pi_u(x_u) \\ &\quad + \sum_{x_v \in \mathbf{X}_v} \sum_{x_u \in \mathbf{X}_u} (x_v - \mu_{\tilde{\pi}_v})(x_u - \mu_{\tilde{\pi}_u})g(v)\beta_{v,u}g(u) \\ &= E[X_v - \mu_{\tilde{\pi}_v}][X_u - \mu_{\tilde{\pi}_u}] \\ &\quad + \beta_{v,u} \sum_{x_v \in \mathbf{X}_v} (x_v - \mu_{\tilde{\pi}_v})g(v) \sum_{x_u \in \mathbf{X}_u} (x_u - \mu_{\tilde{\pi}_u})g(u) \\ &= E[X_v - \mu_{\tilde{\pi}_v}][X_u - \mu_{\tilde{\pi}_u}] + \beta_{v,u}, \qquad (4.6) \end{aligned}$$

$$cov(X_v, X_u) = \beta_{v,u} \qquad (4.7)$$

$\square$

4.B.3 Algorithm for Simulating Random Fields

Now, simulating the random field becomes a simple matter of computing the required probabilities and generating uniform random numbers to select a value for each vertex. For $v \in V$, $d_v \in \mathbb{N}$ is the cardinality of $\mathbf{X}_v$, we denote $\mathbf{X}_v = \{x_v^1, ..., x_v^{d_v}\}$.

Do for $i = 1, \ldots, N$:

1. Based on (4.1), compute

$$\begin{aligned}\Pi(X_{v_{i-1}} = x_{v_{i-1}}, ..., X_{v_1} = x_{v_1}) \\ &= \prod_{k=1}^{i-1} \Pi(X_{v_k} = x_{v_k} | X_{v_{k-1}} = x_{v_{k-1}}, ..., X_{v_1} = x_{v_1}) \\ &= \prod_{k=1}^{i-1} \Pi(X_{v_k} = x_{v_k} | X_{\mathrm{pa}(v_k)} = x_{\mathrm{pa}(v_k)})\end{aligned}$$

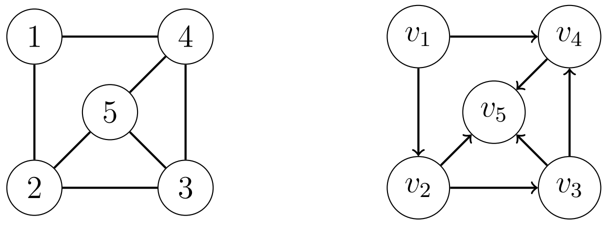
   for all chosen combinations of x_{v_1}, ..., $x_{v_{i-1}}$. Here we choose x_{v_1}, ..., $x_{v_{i-1}}$ as follows: for each $1 \leq k \leq i-1$, if $v_k \in \mathrm{pa}(v_i)$, we use the simulated x_{v_k}; otherwise, we enumerate $x_{v_k} \in \mathbf{X}_{v_k}$.

2. Take marginal to get $\Pi(X_{\mathrm{pa}(v_i)} = x_{\mathrm{pa}(v_i)})$:

$$\Pi(X_{\mathrm{pa}(v_i)} = x_{\mathrm{pa}(v_i)}) = \sum_{v_k \notin \mathrm{pa}(v_i), 1 \leq k \leq i-1} \Pi(X_{v_{i-1}} = x_{v_{i-1}}, ..., X_{v_1} = x_{v_1}).$$

3. Based on $\Pi(X_{\mathrm{pa}(v_i)} = x_{\mathrm{pa}(v_i)})$, compute $\Pi(X_{v_i} = x_{v_i}^j \,\big|\, X_{\mathrm{pa}(v_i)} = x_{\mathrm{pa}(v_i)})$ for $1 \leq j \leq d_{v_i}$, using (4.3).

4. Generate a $[0, 1]$-uniform random variable U. For the given U, there



(a) Undirected Graph G (b) Directed Graph D

Figure 4.1: Graph Construction Example

exists unique $1 \leq j \leq d_{v_i}$ such that

$$\sum_{u=1}^{j-1} \Pi(X_{v_i} = x_{v_i}^u \,|\, X_{\mathrm{pa}(v_i)} = x_{\mathrm{pa}(v_i)}) \leq U < \\ \sum_{u=1}^{j} \Pi(X_{v_i} = x_{v_i}^u \,|\, X_{\mathrm{pa}(v_i)} = x_{\mathrm{pa}(v_i)}).$$

Then set $X_{v_i} = x_{v_i}^j$. For notational convenience, we suppress superscript j and use x_{v_i} to indicate the simulated value $x_{v_i}^j$ of X_{v_i}.

4.C Examples

Example 4.C.1. Suppose we have random variables and pairwise covariances such that our undirected graph $G = (V, E)$ is as given in Figure 4.1a. Then, our procedure for constructing the directed graph, $D = (V, A)$, could result in the graph shown in Figure 4.1b.

Let the common space of states for each vertex be $\mathbf{X} = \{-1, 0, 1\}$. To construct a probability measure Π on $\mathcal{X} = \{-1, 0, 1\}^5$, we use Proposition 4.B.1 to compute the conditional probabilities $\Pi(X_{v_i} = x_{v_i} | X_{\mathrm{pa}(v_i)} = x_{\mathrm{pa}(v_i)})$ for $1 \leq i \leq 5$. To compute, for example, $\Pi(X_{v_4} = x_{v_4} | X_{\mathrm{pa}(v_4)} = x_{\mathrm{pa}(v_4)})$, we have to compute $\Pi(X_{\mathrm{pa}(v_4)} = x_{\mathrm{pa}(v_4)})$ as a prerequisite, using Proposition

4.B.1. So we have

$$\begin{aligned}\Pi(X_{\mathrm{pa}(v_4)} = x_{\mathrm{pa}(v_4)}) &= \Pi(X_{v_3} = x_{v_3}, X_{v_1} = x_{v_1}) \\ &= \sum_{x_{v_2} \in \{-1,0,1\}} \Pi(X_{v_3} = x_{v_3}, X_{v_2} = x_{v_2}, X_{v_1} = x_{v_1}) \\ &= \sum_{x_{v_2} \in \{-1,0,1\}} \big[\Pi(X_{v_3} = x_{v_3} | X_{v_2} = x_{v_2}) \\ &\qquad \times \Pi(X_{v_2} = x_{v_2} | X_{v_1} = x_{v_1}) \Pi(X_{v_1} = x_{v_1})\big].\end{aligned}$$

□

Example 4.C.2. Suppose we are given an image (i.e., a two dimensional grid of pixels) of size $M \times N$. We can construct an undirected graph representing this image using a vertex for each pixel, labeled with the pixel's coordinates in the image (e.g., $(1,2)$). Furthermore, for each pair of pixels which are considered nearby according to the Euclidean distance between them, we can add an edge between the corresponding vertices. Given a set of sample images, we can estimate both the marginal probabilities for each vertex and pairwise covariances for each edge, and can then simulate new images.

Example 4.C.3. Suppose we are given an undirected graph $G = (V, E)$ where the vertices are divided into a known part H^C and an unknown part H, (i.e., V is partitioned into H and H^C). In this case, our goal is to simulate H while using the information contained in H^C. For example, we want to restore a portion of an image given some known set of pixels. We want to order the vertices so that $V = \{v_1, v_2, \dots, v_k, v_{k+1}, \dots, v_n\}$, where $H^C = \{v_1, v_2, \dots, v_k\}$ and $H = \{v_{k+1}, \dots, v_n\}$. This allows us to start simulating immediately from v_{k+1} since all preceding vertices are already known. To order the vertices in this way, we begin by working with G_{H^C}, the subgraph induced by H^C; we then

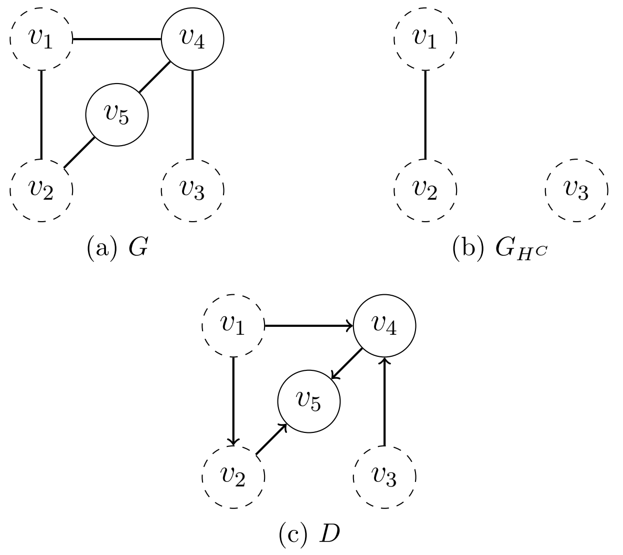


Figure 4.2: Known and Unknown Vertices Example

divide G_{H^C} into its N connected components $G^1_{H^C}, G^2_{H^C}, \ldots, G^N_{H^C}$, and apply the algorithm in Section 4.B.1 for each connected component to obtain directed subgraphs $D^i = (V^i, A^i)$ for $i = 1, \ldots, N$, where $V^i = \{v^i_1, \ldots, v^i_{n_i}\}$. Finally, we apply the algorithm in Section 4.B.1 to G with one slight difference: instead of initializing $V_0 = \emptyset, A_0 = \emptyset$, we initialize $V_0 = \cup_{i=1}^N V^i$ and $A_0 = \cup_{i=1}^N A^i$ so that $V_0 = \{v^1_1, \ldots, v^1_{n_1}, v^2_1, \ldots, v^N_{n_N}\}$. Then, we've ordered the vertices so that $H^C = \{v_1, v_2, \ldots, v_k\}$ and $H = \{v_{k+1}, \ldots, v_n\}$, as desired, and we can begin simulating from v_{k+1}.

For example, consider Figure 4.2a which shows G with vertices from H^C with dashed outlines and vertices from H with solid outlines. The subgraph G_{H^C} is shown in Figure 4.2b, and in this case there are two connected components $G^1_{H^C} = \{v_1, v_2\}$ and $G^2_{H^C} = \{v_3\}$. Figure 4.2c shows the final directed graph $D = (V, A)$.

Chapter 5

Conclusion

In this thesis, I presented three papers prepared during my Masters of Science program at the University of Alberta. In these papers, we developed, implemented, and empirically tested the KNW-CAPTCHA, a novel mechanism for generating CAPTCHAs via random field simulation. We established that the KNW-CAPTCHA is highly readable to humans via a volunteer study as well as results collected via Amazon Mechanical Turk, and we established that it is resistant to automated attacks via multiple OCR programs and a targeted segmentation attack.

Furthermore, we established the general flexibility of our method by describing it in the context of a template design pattern, and described the straightforward process of developing and implementing variants. We performed an in-depth analysis of one such variant, the grey-level KNW-CAPTCHA, and empirically optimized a cost function over several possible parameter sets; we determined that the addition of a grey level provided a useful mechanism for further separating human and computer agents.

Finally, we laid out the algorithm for simulating these correlated random fields in detail, generalized from prior applications in two dimensions to multiple dimensions in the context of graph theory.

Future work in this area would focus on two fronts: further development of the KNW-CAPTCHA variants and the method of selecting them; and further applications of the random field given in Chapter 4. For example, in Chapter 3, estimation of the attack resistance if based on the performance of OCR programs. However, OCR programs do not properly capture the threat of a targeted attack, which may exploit particular weaknesses in a CAPTCHA design. Indeed, one could extend the estimation of attack resistance to resisting all attacks from a catalog of programs consisting of both OCR programs and targeted attacks (where targeted attacks are possibly submitted by other researchers in this area). Selecting a CAPTCHA for deployment could become an automated process which dynamically adjusts to the best known attacks.

The KNW random field lends itself to other applications where the values are discrete and marginal probabilities and pairwise covariances capture the meaningful structure of a problem, including image restoration and OCR. For example, one could invert the CAPTCHA simulation problem and instead implement a CAPTCHA recognition algorithm based on calculating the probabilities of an image under different parameters, where the parameters represent different letters or words; i.e., one would want to select the parameters which maximize the likelihood of the data, as in maximum likelihood estimation. Indeed, one could implement such an attack in included it in the catalog of attack algorithms for automatic CAPTCHA selection.

Bibliography

[1] E. Bursztein, S. Bethard, C. Fabry, J. Mitchell, and D. Jurafsky, "How good are humans at solving captchas? a large scale evaluation," in *Proceedings of the 2010 IEEE Symposium on Security and Privacy*, ser. SP '10, IEEE. Washington, DC, USA: IEEE Computer Society, 2010, pp. 399–413.

[2] G. Mori and J. Malik, "Recognizing objects in adversarial clutter: Breaking a visual CAPTCHA," *IEEE Conference on Computer Vision and Pattern Recognition*, vol. 1, pp. 134–141, 2003.

[3] M. Kouritzin, F. Newton, and B. Wu, "On Random Field CAPTCHA Generation," *Accepted with Mandatory Minor Revisions to IEEE Transactions on Image Processing.*

[4] F. Newton and M. Kouritzin, "On grey levels in random captcha generation," in *Proceedings of SPIE Visual Information Processing XX*, vol. 8056, 2011, pp. 80 560U–80 560U–12.

[5] M. Kouritzin, F. Newton, and B. Wu, "The KNW Random Field," *In Final Preparation.*

[6] M. Blum, L. Von Ahn, J. Langford, and N. Hopper, "The CAPTCHA Project, "Completely Automatic Public Turing Test to tell Computers and Humans Apart"," *Dept. of Computer Science, Carnegie-Mellon Univ., http://www. captcha. net.*

[7] L. von Ahn, M. Blum, N. Hopper, and J. Langford, "CAPTCHA: Using hard AI problems for security," in *Proceedings of Eurocrypt, 2003.* Springer-Verlag, 2003, pp. 294–311.

[8] R. Casey and E. Lecolinet, "A survey of methods and strategies in character segmentation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 18, no. 7, pp. 690–706, 1996.

[9] J. Yan and A. El Ahmad, “A Low-cost Attack on a Microsoft CAPTCHA,” in *Proc. 15th ACM Conf. Computer and Communications Security (CCS 08).* ACM Press, 2008, pp. 543–554.

[10] R. Datta, J. Li, and J. Wang, “Imagination: A robust image-based captcha generation system,” in *Proceedings of the 13th annual ACM international conference on Multimedia.* ACM, 2005, pp. 331–334.

[11] K. Kluever, “Breaking the PayPal HIP: A Comparison of classifiers,” Rochester Institute of Technology Document and Pattern Recognition Lab, https://ritdml.rit.edu/handle/1850/7813, 2008.

[12] G. Moy, N. Jones, C. Harkless, and R. Potter, “Distortion estimation techniques in solving visual captchas,” *Conference on Computer Vision and Pattern Recognition*, vol. 2, pp. 23–28, 2004.

[13] A. Coates, H. Baird, and R. Faternan, “Pessimal print: a reverse turing test,” in *Document Analysis and Recognition, 2001. Proceedings. Sixth International Conference on.* IEEE, 2001, pp. 1154–1158.

[14] M. Chew and H. Baird, “Baffletext: a human interactive proof,” in *In Proceedings of SPIE-IS&T Electronic Imaging, Document Recognition and Retrieval*, January 2003, pp. 305–316.

[15] H. Baird and T. Riopka, “ScatterType: A reading CAPTCHA resistant to segmentation attack,” in *Proc. SPIE*, vol. 5676, no. 1. Citeseer, 2005, pp. 197–201.

[16] J. Yan and A. El Ahmad, “Breaking Visual CAPTCHAs with Naive Pattern Recognition Algorithms,” in *Proc. of the 23rd Annual Computer Security Applications Conference (ACSAC’07).* IEEE computer society, Florida, USA, Dec 2007, pp. 279–291.

[17] M. Kouritzin, F. Newton, and B. Wu, “Properties of the KNW Random Field,” *In Preparation.*

[18] ImageMagick Studio LLC. ImageMagick. http://www.imagemagick.org/.

[19] R. Smith, “An overview of the Tesseract OCR engine,” in *Int. Conf. on Document Analysis and Recognition (ICDAR)*, vol. 2, Curitiba, Brazil, 2007, pp. 629–633.

[20] “Amazon mechanical turk,” http://www.mturk.com/.

[21] A. Kittur, E. Chi, and B. Suh, “Crowdsourcing user studies with mechanical turk,” in *Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems.* ACM, 2008, pp. 453–456.

[22] A. Sorokin and D. Forsyth, “Utility data annotation with amazon mechanical turk,” in *First IEEE Workshop on Internet Vision, CVPR*, 2008, pp. 1–8.

[23] J. Ross, L. Irani, M. Silberman, A. Zaldivar, and B. Tomlinson, “Who are the crowdworkers?: shifting demographics in mechanical turk,” in *Proceedings of the 28th of the international conference extended abstracts on Human factors in computing systems*. New York, NY, USA: ACM, 2010, pp. 2863–2872.

[24] L. Von Ahn, B. Maurer, C. McMillen, D. Abraham, and M. Blum, “reCAPTCHA: Human-Based Character Recognition via Web Security Measures,” *Science*, vol. 321, no. 5895, pp. 1465–1468, September, 2008.

[25] K. Chellapilla, K. Larson, P. Simard, and M. Czerwinski, “Computers beat humans at single character recognition in reading based human interaction proofs (HIPs),” in *Proceedings of the Second Conference on Email and Anti-Spam*. Citeseer, 2005, pp. 21–22.

[26] S. Tsujimoto and H. Asada, “Major components of a complete text reading system,” *Proceedings of the IEEE*, vol. 80, no. 7, pp. 1133–1149, 1992.

[27] R. Hoffman and J. McCullough, “Segmentation methods for recognition of machine-printed characters,” *IBM Journal of Research and Development*, vol. 15, no. 2, pp. 153–165, 1971.

[28] E. Gamma, R. Helm, R. Johnson, J. Vlissides *et al.*, *Design patterns*. Addison-Wesley Reading, MA, 2002, vol. 1.