Syndrome Trellis Codes for sampling: extensions of the Viterbi algorithm

3^{rd} year project report, Tamio-Vesa Nakajima,

Honour School of Computer Science - Part B,

Submitted as part of an MCompSci in Computer Science

Trinity term, 2020

Abstract

We propose a novel algorithm that combines two disparate parts of steganography. One thread of steganography research has focused on modifying an image as little as possible so that it hides the message, using syndrome trellis codes to hide that message. Another thread of research attempts to sample from the distribution of stegotexts, conditioning that they hide the message. Combining these threads, our algorithm efficiently samples stegotexts according to a Markov model with memory, conditioning that they hide the message, using a syndrome trellis code to hide it. We propose a framework that formalises such models, making them usable in the algorithm. A modification of the algorithm is also devised that finds the stegotext with maximal likelihood. We show that the previous steganographic uses of the Viterbi algorithm are a special case of this algorithm.

Contents

Chapter 1

Introduction

Steganography is the procedure of sending hidden information through a public channel, so that it is difficult to detect that hidden information was sent. The simplest setting for it is the *prisoners' problem*, created by Simmons [14]. Suppose Alice and Bob, after exchanging a private key, are separated, and allowed to communicate only through a channel monitored by a warden. How can Alice send Bob messages, hiding them inside stegotexts that the warden deems innocuous?

To illustrate, suppose you are a secret agent. Your spying has discovered whether the enemy will attack by land or by sea, and you need to transmit this information to your handler. This is the secret information. The public channel is a social media account, where you can post images. The private key you and your handler share is a date and time, at which you will post an image. If the sum of the bits in the image is even, then the enemy will attack by land; otherwise, they will attack by sea. The enemy is already suspicious of you, and monitors everything you post. Therefore you would be found out immediately if the image you post is suspicious (for example, an image with a single pixel). Since the number of possible messages is small, we can simply capture innocuous photographs until we find one that hides the correct information. Since the images are drawn from a legitimate source, we assume that they will not seem suspicious to the enemy. Repeatedly sampling innocuous stegotexts until they happen to hide the required message in this way constitutes a simple steganographic protocol: a rejection sampler. Of course, the expected number of samples needed is *exponential* with regards to amount of information in the message.

Rather than trying to sample images, much research has focused on modifying a given legitimate image. After specifying a distortion function (such as UNIWARD [18] or WOW [8]) that quantifies how detectable image modifications are, we attempt to modify the legitimate image so that it hides the message, while also minimising the distortion. A popular approach [15] is to use a linear approximation of the distortion function (one that assigns each one pixel modification the distortion it would cause if no other modifications were made), and then to hide each message bit in the modulo 2 sum of certain subsets of pixels[1]. Such a message encoding scheme is called a *linear code*, since it computes the hidden message by multiplying the image (seen as a vector) with a matrix, modulo 2. Unfortunately, the decision version of this optimisation problem is NP-HARD (it is a generalisation of the COSET WEIGHTS problem [5]). To make the problem tractable we add a *bandedness* constraint, enforcing that each successive message bit depends on pixels that are successively further along in the image (after putting the pixels in some fixed order). Such an encoding scheme is called a *syndrome trellis code*. Then, the *Viterbi algorithm* [19] can be used to find the best modifications in parametrised polynomial time.

These approaches are not perfect: it is not necessarily true that a slightly modified legitimate image is not suspicious. That small changes are not necessarily detectable is refuted, for example, in the case of color images. Many color digital cameras use an array of light sensors, one for each pixel, with a color filter array placed in front of the sensors [11]. This is arranged so that each sensor will receive information for only one color channel in its pixel. The missing color channels must be deduced from neighbouring pixels in a process called *demosaicing*. As has been pointed out previously [2], this introduces linear constraints between the pixels of the photograph. Thus certain photographs, even if they can be represented as bitmaps, cannot be generated by a camera. If we naively modify a few pixels of a photograph, we can easily create images that could not have been made by a camera.

Turning our attention from image covers to textual covers, we see another strand of research. This tries to emulate the desirable properties of the rejection sampler: that the transmitted text is drawn from the distribution of legitimate covers, conditioning that it

[1]The alert reader will note that, for such an encoding to work, there cannot be more message bits than pixels.

hides the message. The most popular approach is an extension of the rejection sampler [1]. It constructs the stegotext in chunks (for example, sentences). Each chunk hides one bit of the message. The transmitted text is built one chunk at a time, from left to right, with each chunk being sampled until it hides the required bit. A bound on the number of samples for each chunk can also be imposed [9] – but this makes it possible that the method will fail. If such a bound is not imposed, this approach runs into the same problem as the rejection sampler: when we try to embed more information in each chunk, the time complexity becomes exponential. Another problem is that we can, in principle, get into a dead end, where there is no way of continuing the text that is compatible with the message. This will happen when there are strong dependencies between chunks (for example, if the chunks are single words rather than sentences), or if we map chunks to bits inappropriately (for example, if we map all chunks to 0).

In summary, the approach outlined in the previous paragraph has three fundamental problems. The first problem is the trade-off between efficiency, completeness and capacity, which seems difficult to resolve in a satisfactory way. The second problem is that it uses an inefficient encoding scheme. Each chunk is responsible for hiding one bit of the message. It seems better to disperse responsibility for hiding each message bit among several chunks, as was done in the image case – since in this way the stegotext seems to depend less on the message. Finally, it is possible to get stuck in dead ends.

This work will fix these problems, using ideas from image steganography. We give an efficient algorithm that samples a stegotext from a Markov chain with memory, conditioning that it hides a message under a syndrome trellis code. The algorithm is guaranteed to generate an answer if one exists. As far as we are aware, this is the first time this has been done. The algorithm can also be modified to find the most likely stegotext. By a reduction argument, we will show that this modified version of the algorithm includes the steganographic uses of the Viterbi algorithm as a special case.

Chapter 2

Background

2.1 Stegosystem

First, we describe the stegosystem we propose: the framework in which our algorithms will function. Alice must communicate a secret *message* with Bob, over a public channel monitored by a warden. The text sent over the channel is called the *stegotext*. The message is a sequence of symbols in $\mathbb{B}$, and the stegotext is a sequence of symbols in Σ. Both have fixed length. In order to make the warden's monitoring meaningful, all participants will share an understanding of what kinds of messages would usually be transmitted over the public channel. These innocent messages will be called *covers*, and this understanding will be codified in a *cover model*. These models will contain a probability distribution over the possible covers. Alice and Bob have shared a private key, from which they will create a *syndrome trellis code*. Together with an *encoding function* this will allow them to recover the message from the stegotext. Alice will then generate a stegotext from which her chosen message can be recovered, and transmit that.

What will be the goal when generating this stegotext? We propose the following ideas

- Sample a stegotext according to the cover model, conditioning that it hides the message.
- Find the most likely stegotext, according to the cover model, that hides the message.

We will devise algorithms to solve both of these problems. These algorithms will use a mathematical object called a *trellis graph*.

Each of the subsequent chapters will explain one of the concepts mentioned earlier in more detail, until they are all finally applied. The *Background* chapter will delineate the notation used, and will properly define syndrome trellis codes. The *Proposed system* chapter will define cover models, the trellis graph, and will describe the algorithms. The *Applications* chapter will show how these algorithms may be applied.

One application is sampling a natural language stegotext. This is the more novel application: it is the first time syndrome trellis codes have been used for sampling (as far as we are aware). This example illustrates how to use the algorithms to sample a stegotext from a Markov chain, conditioning that it encodes a message. Another way to use the algorithm is as a generalised version of the Viterbi algorithm, in the context of image steganography. Whereas, when applied in steganography, the Viterbi algorithm finds the minimum distortion stegotext with respect to some linear distortion function, our algorithm is able to do this with nonlinear distortion functions, when the nonlinearity is restricted based on proximity. As a case study, we apply it to a nonlinear approximation of the UNIWARD cost function. Unfortunately, due to the nature of the distortion function, the algorithm does not outperform traditional methods in image steganography in this case. Nonetheless we believe it is a good illustration of the capabilities of the algorithm. We also show how to reduce the Viterbi algorithm to a special case of our algorithm. We conclude with a few considerations and remarks

2.2 Notation

The following notation will be used.

Sets of sequences. For any set A and any non-negative integer n let A^n represent the set of sequences built from n elements of A; let $A^{<n} = \bigcup_{i=0}^{i<n} A^i$; and let $A^{\leq n} = A^{<n+1}$.

Sequence creation. Let $\langle s_0, \ldots, s_{n-1} \rangle$ denote the sequence whose elements are $s_0, \ldots, s_{n-1}$. Let $\langle f(x) : x \leftarrow \langle s_0, \ldots, s_{n-1} \rangle \rangle$ denote $\langle f(s_0), \ldots, f(s_{n-1}) \rangle$, in that order. For two sequences s and s', let $s +\!\!+ s'$ denote s concatenated with s'. Let x^n represent the length n sequence containing only x. Moreover, we consider that, for the purposes of matrix multiplication, sequences are column vectors.

Special sequences. Let ϵ be the empty sequence. Let $[i, j]$ represent $\langle i, i+1, \ldots, j \rangle$

if $i \leq j$, or ϵ otherwise. Let $[i, j) = [i, j-1]$, and $(i, j] = [i+1, j]$.

Sequence indexing. If $s = \langle s_0, \ldots, s_{n-1} \rangle$, then $s[i] = s_i$. Thus, all sequences are 0-indexed. These conventions also apply to matrices, so $A[i][j]$ is the element on the row i and column j of A, where these are counted from 0. Moreover, let $s[i, j]$ be $\langle s[x] : x \leftarrow [i, j] \rangle$, and define $s[i, j)$ and $s(i, j]$ similarly. Extend this notation to matrices, letting $A[i, j][k, l]$ be the submatrix of A that contains rows $i, \ldots, j$ and columns $k, \ldots, l$; and define $A[i, j)[k, l)$, etc. analogously. Finally, introduce a special value $\perp$; undefined elements of a sequence, such as $s[-1]$, are usually assigned this value.

Pseudocode conventions. We use = for assignment in pseudocode. For a dictionary d, *keys*(d) represents the set of keys of that dictionary. The value that corresponds to key k in dictionary d is $d[k]$. Dictionaries are also *curried*, that is, if a dictionary d has keys from set $A \times B$, then for any $a \in A$, $d[a]$ is a dictionary with keys in B, which contains key-value pair (b, v) if and only if d contained key-value pair $((a, b), v)$.

Miscellaneous notation. The size of a sequence s is given by $|s|$. The alphabet of the message which we will hide will be denoted by $\mathbb{B}$, which we assume to be of form $[0, k)$. Addition and multiplication on $\mathbb{B}$ is done modulo k – including matrix addition and multiplication when matrix elements are in $\mathbb{B}$. Let $\mathbb{P}(x)$ denote the probability of an event x. Let 2^A represent the power set of A. The symbol $*$ is a *wild card* used to indicate information that we don't care about; for instance, if p is a pair and we write "let $(x, *) = p$", then the second element of p is ignored. $\mathbb{R}$ denotes the set of real numbers, and $\mathbb{N}$ denotes the set of natural numbers.

In principle half open intervals of form $s[i, j)$ will be preferred, as is recommend by Dijkstra [4].

2.3 Syndrome Trellis Codes

We now define syndrome trellis codes. These are linear codes (i.e. they can be decoded by multiplying with a matrix), that satisfy a *bandedness* constraint. A parameter called h represents how strict this constraint is. Additional constraints are added so that the codes can encode all possible messages. These codes can be designed in sophisticated ways so as to optimise their efficiency. However, we ignore these considerations and focus on the

requirements that are needed for the algorithms we present.

Definition 1 (Syndrome Trellis Code)**.** A matrix $H \in \mathbb{B}^{M \times N}$, where $M \leq N$, represents a *syndrome trellis code* with constraint length h if and only if there exist sequences *fst*, $lst \in \mathbb{N}^N$ so that:

1. $\forall j \in [0, N) \cdot 0 \leq fst[j] \leq lst[j] \leq M$.
2. $\forall j \in [0, N) \cdot lst[j] - fst[j] < h$.
3. $\forall j \in [0, N-1) \cdot fst[j] \leq fst[j+1] \wedge lst[j] \leq lst[j+1]$.
4. $\forall i \in [0, M), j \in [0, N) \cdot H[i][j] \neq 0 \rightarrow i \in [fst[j], lst[j])$.
5. $\forall i \in [0, M) \cdot \exists j \in [0, N) \cdot H[i][j] \neq 0$.

Let $fst[-1] = lst[-1] = 0$, $fst[N] = lst[N] = M$. For $i \in [0, N)$, define the auxiliary sequences *effect*, Δfst, Δlst, *height* by:

$$\begin{aligned} effect[j] &= \langle H[i][j] : i \leftarrow [fst[j], lst[j]) \rangle \\ \Delta fst[j] &= fst[j] - fst[j-1] \\ \Delta lst[j] &= lst[j] - lst[j-1] \\ height[j] &= lst[j] - fst[j]. \end{aligned}$$

Remark. H represents a code that hides a length M message in a length N sequence. It assigns sequence $s \in \mathbb{B}^N$ the message $H \times s$. We require that $M \leq N$ since it is clearly impossible to hide more than N units of information in a sequence that contains N units of information overall. In practice it is best for $M \ll N$, because this gives us more freedom when creating a sequence that hides the message – since the messages are assigned to more sequences.

The code is built so that the j^{th} element of the sequence only affects range $[fst[j], lst[j])$ of the message. The bandedness constraint, required to make the problems from before tractable, is enforced by bounding the size of the aforementioned ranges, and by enforcing that both their left and right endpoints increase together with j. The conditions mentioned in the definition reflect this:

1. The first condition enforces that $[fst[j], lst[j])$ is a sensible range.

2. The second condition enforces the range size requirement.

3. The third condition enforces that the endpoints increase together with j.

4. The fourth condition enforces that the j^{th} element of the sequence can only effect range $[fst[j], lst[j])$ in the message.

5. The final condition enforces that each message symbol can be affected by the sequence – otherwise, it would not be possible to encode all messages!

The values of *fst* and *lst* at -1 and N will allow us to elegantly treat edge cases later. Finally, *effect*$[i]$ represents the effect that the j^{th} input symbol has on message range $[fst[j], lst[j])$. Note also that *fst*, *lst*, and *effect* fully specify the trellis, using $O(Nh)$ space.

Example. Take $\mathbb{B} = \{0, 1\}$. H is a syndrome trellis code with constraint length 3 if

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

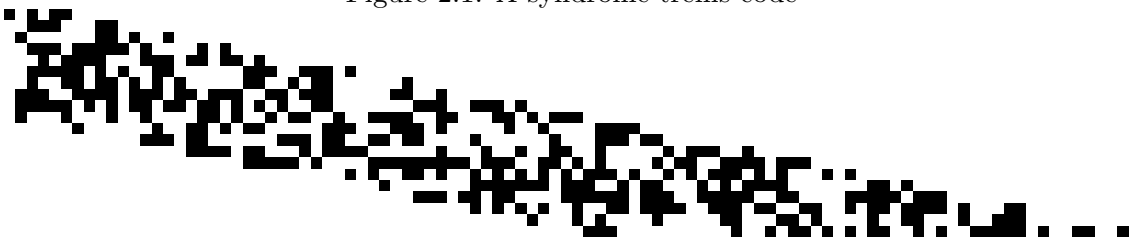
Since it has a null row, H' is not a syndrome trellis code when

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

Example. Now, a larger example. Take $\mathbb{B} = \{0, 1\}$ again. Figure 2.1 represents a syndrome trellis code that maps a sequence of length 100 to a message of length 20, where a black pixel represents a 1 and a white one a 0[1]. Each column is the effect of a sequence symbol on the message; and each row corresponds to a message bit. Thus we can see the constraint length h as being the maximal distance between two 1 values on the same column; and *fst*, *lst* as representing diagonally descending contours, bounding the black pixels above and below. We can thus see why the first 4 conditions hold. The last condition follows from the fact that each row contains a black pixel.

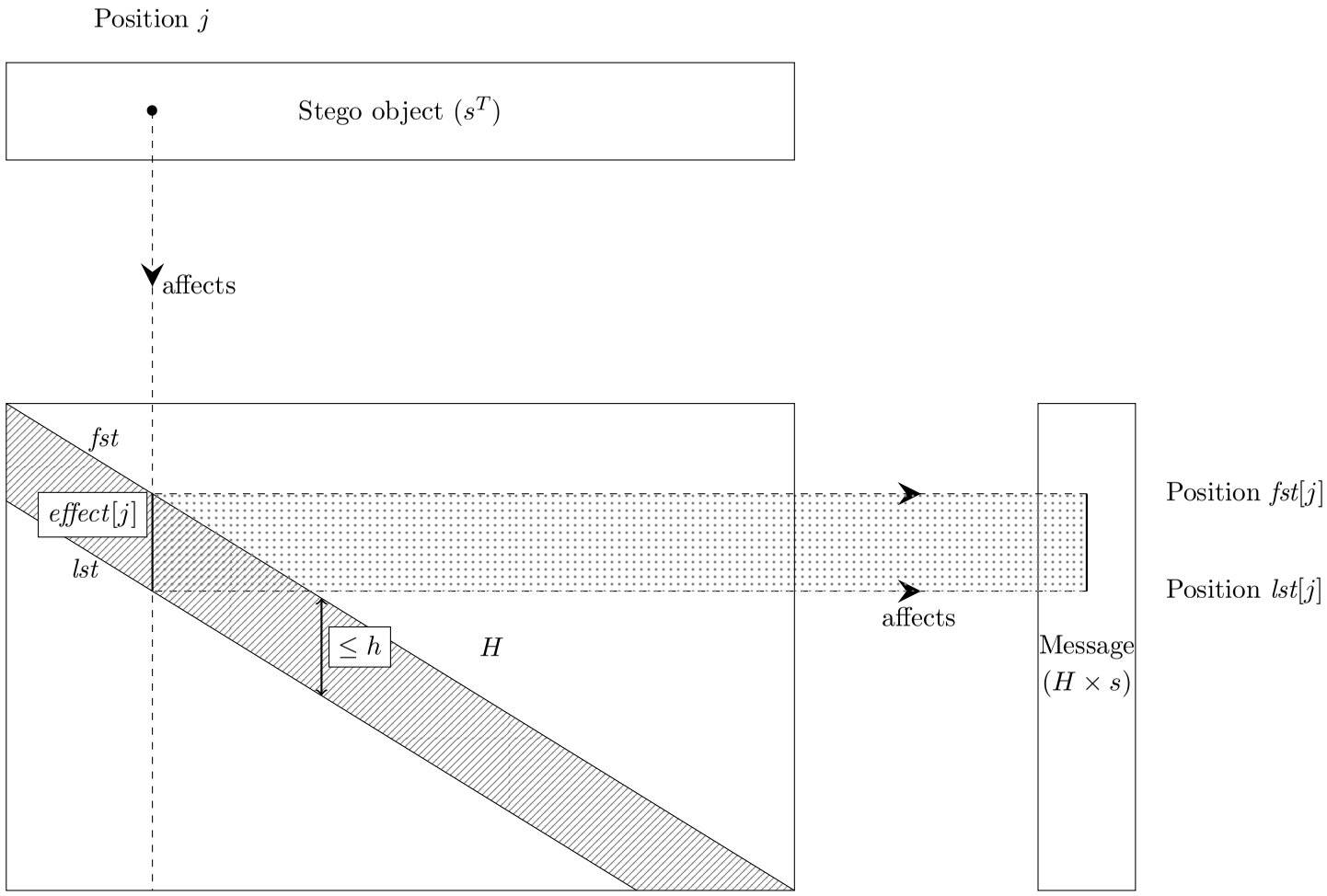
[1]Unfortunately, some PDF viewers mistakenly interpolate the image, making it appear blurry.

Figure 2.1: A syndrome trellis code



Remark. Figure 2.2 shows the relationship between the input sequence s, the syndrome trellis code H, the encoded sequence $H \times s$, the constraint length h, and the auxiliary sequences *fst*, *lst* and *effect*. The white areas of H represent parts of the matrix that contain only 0; whereas the gray hatched area represents parts of the matrix that can contain nonzero values.

Figure 2.2: Syndrome trellis code explanatory diagram



Chapter 3

Proposed system

3.1 Cover models

Now, we focus on modelling covers. Our covers are one-dimensional, and of fixed length. Each symbol in the cover is drawn from a finite alphabet, and depends only on a fixed number of previous symbols. These previous symbols are called *contexts*.

Definition 2 (Cover model)**.** A *cover model* is a 6-tuple $(\Sigma, N, w, \mathbb{C}, enc, p)$, where Σ is finite, $\perp \notin \Sigma$, N and w are natural numbers, $\mathbb{C} \subseteq (\Sigma \cup \{\perp\})^w$, enc is a function from Σ to $\mathbb{B}$, and p is a function from $\Sigma \times \mathbb{C}$ to $\mathbb{R}$. We write $p(c \mid ctxt)$ for $c \in \Sigma$, $ctxt \in \mathbb{C}$. These must satisfy $0 \leq p(c \mid ctxt)$ and $\sum_{c \in \Sigma} p(c \mid ctxt) = 1$ for all $ctxt \in \Sigma^w$. Furthermore define the *candidate function* $cand : \mathbb{C} \to 2^\Sigma$ by

$$cand(ctxt) = \{c \in \Sigma : p(c \mid ctxt) \neq 0\}.$$

With this in mind three further conditions apply to $\mathbb{C}$:

1. $\perp^w \in \mathbb{C}$
2. $ctxt \in \mathbb{C} \wedge c \in cand(ctxt) \to ctxt[1, w) +\!\!+ c \in \mathbb{C}$
3. $ctxt \in \mathbb{C} \to \exists i \in \mathbb{N}, w \in \Sigma^* \cdot ctxt = \perp^i +\!\!+ w$

Remark. We model covers drawn from Σ^N. The model believes that only symbols in $cand(ctxt)$ appear after context $ctxt$ (i.e. when the previous w symbols are equal to $ctxt$). It assigns symbol c appearing after context $ctxt$ probability $p(c \mid ctxt)$. $\mathbb{C}$ is the set of

possible contexts. Finally, $\perp$ represents a context symbol that is before the beginning of a cover. Thus $\langle \perp, \perp, 1, 2 \rangle$ is a context from a sequence beginning with $\langle 1, 2 \rangle$, and $p(x|\langle \perp, \perp, 1, 2 \rangle)$ is the probability that x appears as the third symbol in such a sequence. We adopt this convention, rather than use a *"start of sequence"* symbol, so that all contexts have length w. This is useful in the algorithms that follow.

Definition 3 (Sequence probability, encoding)**.** Extend p and enc to $\Sigma^{\leq N}$, with

$$p(s) = \prod_{i=0}^{i<|s|} p\left(s[i] \mid s[i-w, i)\right)$$

$$enc(s) = \langle enc\left(x\right) : x \leftarrow s \rangle .$$

Remark. p shows us what probability the model assigns to sequences in $\Sigma^{\leq N}$, and enc gives us a way to map Σ^N to $\mathbb{B}^N$. Thus, for a syndrome trellis code H we can use enc to recover a message from stegotext s using the expression

$$H \times enc(s). \tag{3.1}$$

Definition 4 (Admissibility)**.** Say $s \in \Sigma^{\leq N}$ is *admissible* if and only if $p(s) \neq 0$.

Remark. Equivalently, s is admissible if and only if $s[i] \in cand\left(s[i-w, i)\right)$ for all $i \in [0, |s|)$. Thus a sequence is admissible if and only if it could be generated by an algorithm that builds it from left to right, using the candidate function to select symbols.

We restrict our attention to admissible stegotexts. If we were to send an inadmissible stegotext, then it would immediately be deemed suspicious, since such a sequence would never be sent normally.

Definition 5 (Cover model size)**.** Define the size $|C|$ of a cover model C to be

$$\sum_{ctxt \in \mathbb{C}} \left| cand\left(ctxt\right) \right| .$$

Remark. We intend $|C|$ to be an asymptotic bound on the size of the model in memory, under the following assumptions:

- w is a small constant.

- Any symbol in Σ can be stored in a small, constant amount of space.
- p is stored as a dictionary, with keys in $\mathbb{C} \times \Sigma$, and values in $\mathbb{R}$. This dictionary only stores key (ctxt, c) if $c \in \mathit{cand}(\mathit{ctxt})$, since all other keys would be associated with value 0.
- enc takes up a constant amount of space in the representation of C.

We will use this notion of size to express the asymptotic complexity of the algorithms presented later on.

These cover models formalise Markov models with memory, giving us convenient notations and conventions that will be useful further on. As an exercise, we show that p is a well defined probability mass function on Σ^N.

Theorem 1. *For any cover model C, p is a well defined probability mass function on Σ^N.*

Proof. We prove that $p(s) \geq 0$ for $s \in \Sigma^N$, and that $\sum_{s \in \Sigma^N} p(s) = 1$.

For the first part, note that, for $s \in \Sigma^N$

$$\begin{aligned} p(s) &= \prod_{i=0}^{i<N} p\left(s[i] \mid s[i-w, i)\right) && \text{(Defn. } p\text{)} \\ &\geq \prod_{i=0}^{i<N} 0 && \text{(Defn. } p\text{)} \\ &\geq 0. \end{aligned}$$

For the second part, use induction on $k \in [0, N]$ to prove that

$$\sum_{s \in \Sigma^k} \prod_{i=0}^{i<k} p\left(s[i] \mid s[i-w, i)\right) = 1.$$

For the base case, note that

$$\begin{aligned} \sum_{s \in \Sigma^0} \prod_{i=0}^{i<0} p\left(s[i] \mid s[i-w, i)\right) &= \sum_{s \in \Sigma^0} 1 && \text{(the empty product equals 1)} \\ &= 1. && (|\Sigma^0| = 1) \end{aligned}$$

For the inductive step, assume that

$$\sum_{s \in \Sigma^k} \prod_{i=0}^{i<k} p\left(s[i] \mid s[i-w, i)\right) = 1.$$

We want to prove

$$\sum_{s' \in \Sigma^{k+1}} \prod_{i=0}^{i \leq k} p\left(s'[i] \mid s'[i-w, i)\right) = 1.$$

Let $s' = s +\!\!+ c$. Then the left side of the equation becomes

$$\sum_{s \in \Sigma^k} \sum_{c \in \Sigma} \left(p\left(c \mid s[k-w, k)\right) \prod_{i=0}^{i<k} p\left(s[i] \mid s[i-w, i)\right) \right).$$

By distributivity this equals

$$\sum_{s \in \Sigma^k} \left(\sum_{c \in \Sigma} p\left(c \mid s[k-w, k)\right) \right) \left(\prod_{i=0}^{i<k} p\left(s[i] \mid s[i-w, i)\right) \right).$$

By the definition of p, the inner sum is equal to 1, so this becomes

$$\sum_{s \in \Sigma^k} \prod_{i=0}^{i<k} p\left(s[i] \mid s[i-w, i)\right).$$

And this equals 1 by the inductive hypothesis.

Thus p is a probability mass function on Σ^N. □

Example. Consider modelling sequences of length 3, whose symbols are drawn from $[0, 3)$, where the first symbol is 0, and where adjacent symbols are different. All such sequences are considered equally likely. Suppose that $\mathbb{B} = [0, 3)$.

To construct a model that expresses this distribution, set $N = 3$, $w = 1$, $\Sigma = [0, 3)$, $\mathbb{C} = \Sigma \cup \{\perp\}$ and let $enc(c) = c$. Finally set

$$p\left(c \mid s\right) = \begin{cases} \frac{1}{2}, & c = (s \pm 1) \mod 3, s \neq \perp \\ \frac{1}{3}, & s = \perp \\ 0, & \text{otherwise.} \end{cases}$$

Note that the conditions for $\mathbb{C}$ are obviously respected; moreover we show that $0 \leq p\left(c \mid s\right)$

and $\sum_{c\in\Sigma} p\,(c \mid s) = 1$. The first is immediate: $0 \leq \frac{1}{3} \leq \frac{1}{2}$. For the second, note that $p\,(0 \mid \perp) + p\,(1 \mid \perp) + p\,(2 \mid \perp) = \frac{1}{3} + \frac{1}{3} + \frac{1}{3} = 1$, and that $p\,(0 \mid s) + p\,(1 \mid s) + p\,(2 \mid s) = \frac{1}{2} + \frac{1}{2} + 0 = 1$ for $s \in [0, 3)$. So the model is well defined.

The probability distribution described by this model corresponds to the specified one, since all admissible sequences begin with 0, have distinct adjacent symbols, and have equal probability ($\frac{1}{4}$).

Remark. We are now equipped with the notions necessary to rigorously expressed the problems from the *Stegosystem* section. For a given cover model C, a syndrome trellis code H and a message m, we want to:

- Sample an $s \in \Sigma^N$ according to p conditional on $H \times enc(s) = m$.
- Find an $s \in \Sigma^N$ such that $H \times enc(s) = m$ and $p(s)$ is maximal.

3.2 Trellis Graph

Now, we describe the trellis graph. For the following definitions, fix syndrome trellis code H with constraint length h, cover model C, and message m of length M. We construct this graph to succinctly represent the set of admissible stegotexts s that hide m (i.e. for which $H \times enc(s) = m$).

The graph has $N + 1$ layers, indexed from 0 to N, where the vertices in the i^{th} layer represent one or more length i partial stegotexts. Each vertex will contain additional information about its partial stegotexts. If a vertex would correspond to partial stegotexts that cannot be extended so as to encode m and to be admissible, it will be discarded. The edges are directed, each labelled with a symbol (the last symbol of the partial stegotexts of the destination vertex of the edge) and a real number (the probability of selecting that symbol).

The key property that we will prove in this section is that paths that start at the first layer and end at the last one correspond to admissible sequences that hide m (and vice versa). The link between the sequences and the paths is realised through the symbols marking the edges. We will also prove that probability is preserved in this correspondence: the product of the real numbers on the edges of the path is equal to the probability of the

corresponding sequence.

The notion of a trellis graph is not original – it comes from the Viterbi algorithm [19]. The trellis graph described here augments the one used in the Viterbi algorithm with additional information, with each node also storing a context *ctxt* (i.e. the last w symbols in the partial stegotext).

In principle, this graph need not be explicitly constructed to use the algorithms presented later (in fact, we give pseudocode versions of the algorithms that do not). However, formulating these algorithms as graph algorithms makes understanding them easier, makes proving their correctness simpler, and makes analysing their efficiency more straightforward.

Definition 6 (Trellis graph vertex)**.** The vertices of the trellis graph are tuples of form $(i, \mathit{mask}, \mathit{ctxt})$, where $i \in [0, N]$, $\mathit{mask} \in \mathbb{B}^{\mathit{height}[i-1]}$, $\mathit{ctxt} \in \mathbb{C}$. The set of all possible vertices is $\mathbb{V}$.

Remark. Such a vertex represents any length i partial stegotext where the last w symbols equal *ctxt*, and where the hidden message (after 0-padding) is equal to *mask*, in the range where the last chosen symbol can affect the message (i.e. $[\mathit{fst}[i-1], \mathit{lst}[i-1]]$).

Henceforth *mask* will refer to a subsequence of the message hidden in a stegotext.

Definition 7 (Trellis graph edges)**.** The edges of the trellis graph are 4-tuples $(v, v', c, r) \in \mathbb{V} \times \mathbb{V} \times \Sigma \times \mathbb{R}$, written $v \xrightarrow[r]{c} v'$. r is called the *probability* of the edge.

Remark. Each edge is labelled with a symbol in Σ and a probability. The symbol is the one chosen to reach the new vertex; the probability is the probability of such a choice. In particular, an edge that starts from $(i, \mathit{mask}, \mathit{ctxt})$, if labelled with symbol c, should be labelled with probability $p(c \mid \mathit{ctxt})$, and should lead to $(i+1, \mathit{mask}', \mathit{ctxt}')$ where mask' and ctxt' are the mask and context reached after setting the i^{th} stegotext symbol to c. We now show how to compute mask' and ctxt'.

Definition 8 (Next vertex function)**.** For $v = (i, \mathit{mask}, \mathit{ctxt}) \in \mathbb{V}$, where $i < N$, and $c \in \Sigma$,

define

$$ctxt' = ctxt[1, w) +\!\!+\, c$$
$$mask' = \left(mask[\Delta fst[i], height[i-1]) +\!\!+\, 0^{\Delta lst[i]}\right) + enc(c) \times effect[i].$$

Now define the partial function $nextVertex : \mathbb{V} \times \Sigma \to \mathbb{V}$ by

$$nextVertex(v, c) = (i + 1, mask', ctxt').$$

The function is partial since it is not well defined when $i = N$, or when $ctxt' \notin \mathbb{C}$.

Definition 9 (Good vertices)**.** For any vertex $v = (i, mask, ctxt) \in \mathbb{V}$, say that v is a *good* vertex if and only if $m[fst[i-1], fst[i]) = mask[0, \Delta fst[i])$.

Remark. Thus, a vertex is good if the partial stegotexts it represents have correctly set the message symbols that can be changed by the last chosen stegotext symbol, and that cannot be changed by further extending the stegotext. This is a rigorous analogue of the intuitive notion of "vertices that correspond to partial stegotexts that can be extended so as to encode m".

With these notions in mind we can now define the trellis graph.

Definition 10 (Trellis graph)**.** Let the trellis graph (V, E) be the smallest graph such that:

- $v_0 \in V$ where $v_0 = (0, \epsilon, \perp^N)$.
- If $v \in V$, where $v = (i, mask, ctxt)$, and if $c \in cand(ctxt)$ and v' is defined and *good*, where $v' = nextVertex(v, c)$, then $v' \in V$ and $v \xrightarrow[p(c \mid ctxt)]{c} v' \in E$.

Remark. Note that $m[fst[-1], fst[0]) = \epsilon = mask[0, \Delta fst[0])$, so v_0 is *good*. Thus all vertices in V are indeed *good*. Also, note that since at most one edge (and thus at most one node) is added for each way of choosing a cover position (of which there are N), a context followed by a candidate (of which there are $|C|$) and a mask (of which there are $|\mathbb{B}|^h$), $|V| + |E| = O(N|C||\mathbb{B}|^h)$ – a fact which we will prove rigorously later.

Example. Set:

- $\Sigma = \mathbb{B} = \{0, 1\}$.
- $m = \langle 0, 1, 0 \rangle$.
- $N = 5$.
- $w = 1$.
- $p(c \mid s) = \frac{1}{2}$.
- $H = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \end{pmatrix}$.

The resulting trellis graph is shown in figure 3.1.

Remark. Now that we have defined the trellis graph, we will:

- Bound its size, in terms of $N, |C|, |\mathbb{B}|, h$.
- Show that it is directed and acyclic.
- Show that the paths in the graph are in a direct correspondence with the admissible stegotexts that hide m.
- Show that the product of the edge probabilities of a path is equal to the probability of the corresponding sequence according to C.

Thus we will express the problems we want to solve in the language of graphs, in which they become:

- Sample a length N path that starts from v_0, with probability proportional to the product of its edge probabilities.
- Find a length N path that starts from v_0 such that the product of the path's edge probabilities is maximal.

Theorem 2 (Size of a trellis graph)**.**

$$|V| + |E| = O(N|C||\mathbb{B}|^h)$$

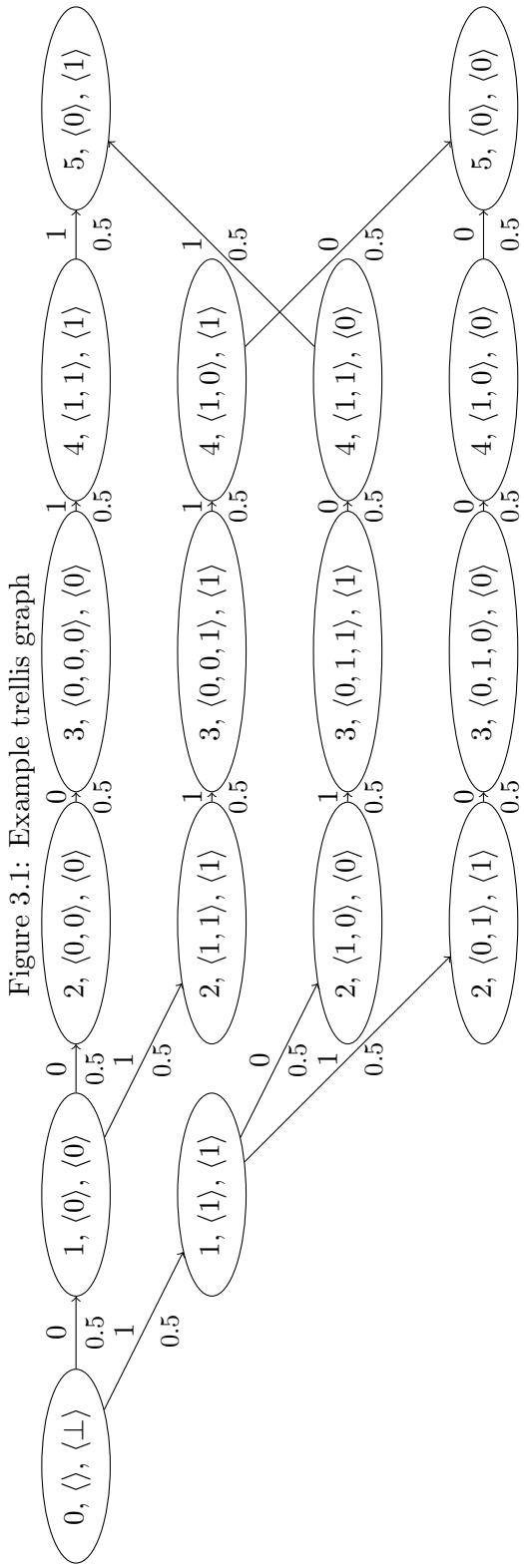


Figure 3.1: Example trellis graph

Proof. First, other than v_0, vertices are only added to the graph at the same time as edges. So $|V| \leq |E| + 1$. Therefore $|V| = O(|E|)$, and we need only prove $|E| = O(N|C||\mathbb{B}|^h)$.

Now, consider any edge $v \xrightarrow[x]{c} v' \in E$ where $v = (i, \mathit{mask}, \mathit{ctxt})$. By construction $x = p(c \mid \mathit{ctxt}) \neq 0$. However, as $p(c \mid \mathit{ctxt})$ is nonzero, $c \in \mathit{cand}(\mathit{ctxt})$. So each such edge corresponds to an element in one of the sets $\mathit{cand}(\mathit{ctxt})$ for some $\mathit{ctxt} \in \mathbb{C}$. Moreover, at most $N|\mathbb{B}|^h$ edges correspond to each element (one for each possible value of i and mask). Since the total number of such elements is $|C|$, it follows that $|E| \leq N|C||\mathbb{B}|^h$, and furthermore $|V| + |E| = O(N|C||\mathbb{B}|^h)$ □

Remark. This fact is important for the efficiency of the algorithms presented later in the paper. It also clearly shows what the main factors in these algorithms' time complexity will be: a linear factor in terms of N, a linear factor in terms of $|C|$ (which can be exponential in terms of w, but less if the model is sparse), and an exponential factor in terms of h.

Theorem 3 (Structure of a trellis graph)**.** *V can be partitioned into layers $V_0, \ldots, V_N$ such that edges starting in any layer always go to the next layer.*

Proof. Define

$$V_i = \{v \in V : v = (i, *, *)\}.$$

Consider any $(i, *, *) \in V_i$ where $(i, *, *) \xrightarrow[*]{*} v' \in E$. By construction, v' is equal to $\mathit{nextVertex}((i, *, *), c)$ for some $c \in \Sigma$. Thus $v' = (i + 1, *, *)$. So $v' \in V_{i+1}$. □

Remark. This means that the trellis graph is directed and acyclic, with an easily constructed topological ordering (i.e. layer order). Also, the length N paths that start from v_0 are precisely those that end in V_N.

The following definitions show how to link admissible sequences that hide m to paths in the graph; and the theorems prove that the link is correct and preserves the probabilities of the paths and sequences. The definitions are simple; the proofs are somewhat technical yet straightforward. We include them for completeness – their results are fairly easy to see from construction.

Definition 11. For any path $v_0 \xrightarrow[*]{s_0} \ldots \xrightarrow[*]{s_{N-1}} v_N$, let the sequence associated with this path be $\langle s_0, \ldots, s_{N-1} \rangle$.

Definition 12. For any sequence $s \in \Sigma^N$, consider the vertices $v_0, \ldots, v_N$, where v_0 has already been defined, and where $v_{i+1} = \mathit{nextVertex}(v_i, s[i])$. If these form a path in the trellis graph, say that it is the path associated with s.

Theorem 4. *If P is a length N path in the trellis graph, then the sequence s associated with P is admissible and hides the message m. Moreover, $p(s)$ is equal to the product of the edge probabilities of P.*

Proof. First note that, due to the construction of the trellis graph, we see that that the path P is of form

$$\begin{aligned}(0, s[-w, 0), \mathit{mask}_0) &\xrightarrow[p(s[0] \mid s[-w,0))]{s[0]} (1, s[1-w, 1), \mathit{mask}_1) \\ &\xrightarrow[p(s[1] \mid s[1-w,1))]{s[1]} \cdots \\ &\xrightarrow[p(s[N-1] \mid s[N-w-1,N-1))]{s[N-1]} (N, s[N-w, N), \mathit{mask}_N).\end{aligned}$$

In other words, the i^{th} vertex (starting from 0) is $(i, s[i-w, i), \mathit{mask}_i)$, and the edge from the i^{th} to the $(i+1)^{th}$ vertex is of form

$$(i, s[i-w, i), \mathit{mask}_i) \xrightarrow[p(s[i] \mid s[i-w,i))]{s[i]} (i+1, s(i-w, i], \mathit{mask}_{i+1}).$$

This shows us that $p(s)$ is equal to the product of the edge probabilities in P. Moreover, since each edge belongs to the graph, none of them are labelled with 0. So $p(s) \neq 0$ and thus s is admissible. What remains to be proven is that s hides the message m.

To see why this is the case, show, by induction on i, that $H \times (\mathit{enc}(s[0, i)) +\!\!+ 0^{N-i}) = m[0, \mathit{fst}[i-1]) +\!\!+ \mathit{mask}_i +\!\!+ 0^{M-\mathit{lst}[i-1]}$. When $i = N$, since $\mathit{lst}[N-1] = M$ (otherwise condition 5 of definition 1 cannot hold), and since, as the last node of P is good, $\mathit{mask}_N = m[\mathit{fst}[N-1], \mathit{lst}[N-1))$, this implies that $H \times \mathit{enc}(s) = m$, as required.

For the base case, when $i = 0$, the required equation becomes $H \times 0^N = 0^M$ (as $\mathit{fst}[-1] = \mathit{lst}[-1] = 0$), which is obvious. For the inductive step, assume, for some $i < N$, that

$$H \times (\mathit{enc}(s[0, i)) +\!\!+ 0^{N-i}) = m[0, \mathit{fst}[i-1]) +\!\!+ \mathit{mask}_i +\!\!+ 0^{M-\mathit{lst}[i-1]}.$$

Now consider the value of

$$H \times (enc(s[0,i]) +\!\!+ 0^{N-i-1}) - H \times (enc(s[0,i)) +\!\!+ 0^{N-i}).$$

Due to distributivity this equals

$$H \times (enc(s[0,i]) +\!\!+ 0^{N-i-1} - enc(s[0,i)) +\!\!+ 0^{N-i}).$$

Factoring out the shared prefix and suffix gives us

$$H \times \left(0^{i} +\!\!+ enc\left(s[i]\right) +\!\!+ 0^{N-i-1}\right).$$

But, the product of a matrix with a sequence with precisely one nonzero entry is a column of the matrix, multiplied by the nonzero entry. The column in question is $\mathit{effect}[i]$ but properly padded with zeroes. Thus the difference is

$$(0^{\mathit{fst}[i]} +\!\!+ \mathit{effect}[i] +\!\!+ 0^{M-\mathit{lst}[i]}) \times enc(s[i]).$$

This implies that $H \times (enc(s[0,i]) +\!\!+ 0^{N-i-1})$ is equal to

$$H \times (enc(s[0,i)) +\!\!+ 0^{N-i}) + (0^{\mathit{fst}[i]} +\!\!+ \mathit{effect}[i] +\!\!+ 0^{M-\mathit{lst}[i]}) \times enc(s[i]).$$

And using the inductive hypothesis, this is just

$$\left(m[0,\mathit{fst}[i-1]) +\!\!+ \mathit{mask}_i +\!\!+ 0^{M-\mathit{lst}[i-1]}\right) + (0^{\mathit{fst}[i]} +\!\!+ \mathit{effect}[i] +\!\!+ 0^{M-\mathit{lst}[i]}) \times enc(s[i]).$$

Split this sequence into 4 chunks, the first of length $\mathit{fst}[i-1]$, the second of length $\mathit{fst}[i] - \mathit{fst}[i-1]$, the third of length $\mathit{lst}[i] - \mathit{fst}[i]$, and the last containing the remaining symbols. Note that:

1. The first contains $m[0,\mathit{fst}[i-1]) + 0^{\mathit{fst}[i-1]} \times enc(s[i])$. This is equal to $m[0,\mathit{fst}[i-1])$.

2. The second contains $\mathit{mask}_i[0,\mathit{fst}[i]-\mathit{fst}[i-1]) + 0^{\mathit{fst}[i]-\mathit{fst}[i-1]} \times enc(s[i])$. Since $\mathit{fst}[i] - \mathit{fst}[i-1] = \Delta\mathit{fst}[i]$, this is equal to $\mathit{mask}_i[0,\Delta\mathit{fst}[i])$. But note that vertex $(i, s[i -$

$w, i), mask_i)$ belongs to the graph, and thus is *good*. So $mask_i[0, \Delta fst[i]) = m[fst[i-1], fst[i])$. These are the contents of the chunk.

3. The third contains

$$(mask_i[fst[i] - fst[i-1], lst[i-1] - fst[i-1]) +\!\!+\, 0^{\Delta lst[i]}) + effect[i] \times enc(s[i]).$$

Note that $fst[i] - fst[i-1] \leq lst[i-1] - fst[i-1]$, since $fst[i] \leq lst[i-1]$, since otherwise condition 5 of definition 1 cannot be fulfilled. The first term of this sum is $mask_i[\Delta fst[i], height[i-1])$, so the contents are $mask_i[\Delta fst[i], height[i-1]) + effect[i] \times enc(s[i])$. Now, consider the edge leading into vertex $(i+1, s(i-w, i], mask_{i+1})$ in path P. Since the edge belongs to the graph, the vertex is constructed using the *nextVertex* function. So $mask_{i+1}$ coincides with the value of this chunk.

4. The final chunk contains only 0.

Therefore the sequence is equal to

$$m[0, fst[i-1]) +\!\!+\, m[fst[i-1], fst[i]) +\!\!+\, mask_{i+1} +\!\!+\, 0^{M-lst[i]}.$$

Joining the first two terms into $m[0, fst[i])$, we conclude that $H \times (enc(s[0, i]) +\!\!+\, 0^{N-i-1})$ is equal to $m[0, fst[i]) +\!\!+\, mask_{i+1} +\!\!+\, 0^{M-lst[i]}$. This completes the inductive step, and the proof. □

Theorem 5. *If s is an admissible sequence that hides the message m, then a length N path P, associated with s, exists in the trellis graph. Moreover, $p(s)$ is equal to the product of the edge probabilities of P.*

Proof. Suppose $s \in \Sigma^N$ such that $H \times enc(s) = m$. By definition the only path P that is associated with s has form $v_0 \xrightarrow[*]{s[0]} \ldots \xrightarrow[*]{s[N-1]} v_N$, where $v_i = nextVertex(v_{i-1}, s[i-1])$ for $i > 0$. It is easy to see that contexts in these vertices are successive length w subsequences of s. So, we note that $v_i = (i, mask_i, s[i-w, i))$, where $mask_i \in \mathbb{B}^{height[i-1]}$, and $mask_0 = \epsilon$. Thus the edge from vertex v_i to vertex v_{i+1} is labelled with probability $p(s[i]|s[i-w, i))$.

This shows us that the product of the edge probabilities of P is $\prod_{i=0}^{N-1} p(s[i]|s[i-w, i))$, which is equal to $p(s)$. Moreover, since s is admissible, and thus $p(s) > 0$, all of these

probabilities are nonzero[1].

We now show that the path appears in the graph. Remember that, if a vertex x is included in the graph, and $y = \mathit{nextVertex}(x, *)$, then y is included in the graph only if the probability of the edge from x to y is nonzero, and if y is *good*. Since we have already shown that all the edge probabilities in P are nonzero, and since v_0 is always included in the graph, all that remains is to show that $v_1, \ldots, v_N$ are *good*. Equivalently, we must show $\mathit{mask}_i[0, \Delta\mathit{fst}[i]) = m[\mathit{fst}[i-1], \mathit{fst}[i])$ for $i \in (0, N]$.

To do this, we will show a stronger claim: that $\mathit{mask}_i[0, \Delta\mathit{fst}[i]) = m[\mathit{fst}[i-1], \mathit{fst}[i])$, and $H \times \left(\mathit{enc}(s)[0, i) +\!\!+\, 0^{N-i}\right) = m[0, \mathit{fst}[i-1]) +\!\!+ \mathit{mask}_i +\!\!+\, 0^{M-\mathit{lst}[i-1]}$ for $i \in [0, N]$, by induction on i.

For the base case, take $i = 0$. Now

$$\begin{aligned}
\mathit{mask}_0[0, \Delta\mathit{fst}[0]) &= \mathit{mask}_0[0, 0) && \text{(defn. } \Delta\mathit{fst}\text{)} \\
&= \epsilon \\
&= m[0, 0) \\
&= m[\mathit{fst}[-1], \mathit{fst}[0]). && \text{(defn. } \mathit{fst}\text{)}
\end{aligned}$$

Moreover

$$\begin{aligned}
H \times \left(\mathit{enc}(s)[0, 0) +\!\!+\, 0^{N-0}\right) &= H \times 0^N \\
&= 0^M \\
&= m[0, 0) +\!\!+ \mathit{mask}_0 +\!\!+\, 0^{M-0} && (m[0, 0) = \mathit{mask}_0 = \epsilon) \\
&= m[0, \mathit{fst}[-1]) +\!\!+ \mathit{mask}_0 +\!\!+\, 0^{M-\mathit{lst}[-1]}. && \text{(defn. } \mathit{fst}\text{, } \mathit{lst}\text{)}
\end{aligned}$$

So the base case holds.

For the inductive step, assume that $\mathit{mask}_i[0, \Delta\mathit{fst}[i]) = m[\mathit{fst}[i-1], \mathit{fst}[i])$ and

$$H \times \left(\mathit{enc}(s)[0, i) +\!\!+\, 0^{N-i}\right) = m[0, \mathit{fst}[i-1]) +\!\!+ \mathit{mask}_i +\!\!+\, 0^{M-\mathit{lst}[i-1]}$$

[1]This also implies that all the contexts mentioned are indeed members of $\mathbb{C}$

for a fixed $i < N$. As in the previous proof, $H \times (enc(s[0,i]) +\!\!+\, 0^{N-i-1})$ is equal to

$$(m[0, fst[i-1]) +\!\!+\, mask_i +\!\!+\, 0^{lst[i-1]}) + (0^{fst[i]} +\!\!+\, effect[i] +\!\!+\, 0^{M-lst[i]}) \times enc(s[i]).$$

By splitting this expression into chunks as in the last proof, it follows that $H \times (enc(s[0,i]) +\!\!+\, 0^{N-i-1})$ is equal to $m[0, fst[i]) +\!\!+\, mask_{i+1} +\!\!+\, 0^{M-lst[i]}$, as required.

To complete the inductive step, we prove that $mask_{i+1}[0, \Delta fst[i+1]) = m[fst[i], fst[i+1])$. To do this, consider

$$H \times enc(s) - H \times (enc(s[0,i]) +\!\!+\, 0^{N-i-1}).$$

By distributivity this is equal to

$$H \times \left(enc(s) - (enc(s[0,i]) +\!\!+\, 0^{N-i-1})\right).$$

Due to the common prefix, this is equal to

$$H \times \left(0^{i+1} +\!\!+\, enc(s)[i+1, N)\right).$$

But this is equal to a linear combination of columns $i+1, \ldots, N-1$ of H. By the structure of H, these can only have nonzero entries on positions $fst[i+1], \ldots, M-1$. Therefore $H \times enc(s)$ and $H \times (enc(s[0,i]) +\!\!+\, 0^{N-i-1})$ coincide until position $fst[i+1]$. In particular, they coincide on positions belonging to the range $[fst[i], fst[i+1])$. But note that $H \times enc(s) = m$ by assumption, so the subsequence of $H \times (enc(s[0,i]) +\!\!+\, 0^{N-i-1})$ corresponding to indices $[fst[i], fst[i+1])$ is equal to $m[fst[i], fst[i+1])$. As we have already shown that $H \times (enc(s[0,i]) +\!\!+\, 0^{N-i-1}) = m[0, fst[i]) +\!\!+\, mask_{i+1} +\!\!+\, 0^{M-lst[i]}$, by considering indices in $[fst[i], fst[i+1])$ we conclude that $m[fst[i], fst[i+1]) = mask_{i+1}[0, \Delta fst[i+1])$, as required. This completes the inductive step, and the proof. □

Remark. These theorems show that the paths in the trellis graph that start at v_0 and end at V_N (i.e. the ones with length N) correspond exactly to the admissible sequences that hide the secret message, and that probability is preserved by the correspondence. Thus, we only need to solve the following problems:

- Sample a path that starts at v_0 and ends at V_N with probability proportional to the product of its edge probabilities.
- Find the path that starts at v_0 and ends at V_N where the product of its edge probabilities is maximal.

3.3 Algorithms

3.3.1 Sampling a stegotext

We now show how to sample a stegotext s that encodes message m according to distribution p. As stated earlier, it is sufficient to sample a path from v_0 to V_N, with probability proportional to the product of the probabilities of the edges in the path. We say "proportional" since these products do not form a probability distribution. An underlying assumption is that such a stegotext (or equivalently, such a path) actually exists. If the model assigns zero probability to the event that the stegotext hides the message, then clearly any stegotext that hides it is immediately suspicious, so it is pointless to try to send the message. We can try to arrange that this by using an encoding function that maps Σ to $\mathbb{B}$ approximately uniformly, and by insuring that the model is not too sparse.

First, for each vertex, we calculate a value that is proportional to the probability of reaching that vertex if the graph is traversed according to edge probabilities. Thus we calculate the sum of the products of the probabilities on the edges of all paths from v_0 to that vertex. In other words, if we let $d[v]$ represent this value for vertex v, we define it by

$$d[v] = \sum_{P \text{ a path from } v_0 \text{ to } v} \left(\prod_{c \text{ an edge probability of } P} c \right).$$

These values can be calculated with recurrence relation

$$d[v] = \sum_{w \xrightarrow[p]{*} v \in E} p d[w].$$

And with base condition $d[v_0] = 1$. Since the trellis graph is a directed acyclic graph, d can be computed in linear time by traversing the graph in topological order, and applying

the recurrence relation.

Now, let the path we sample be $v_0, \ldots, v_N$. We build this path in reverse order. To determine v_N, consider vertices in V_N, and select one with probability proportional to d; that is, select $v \in V_N$ with probability

$$\frac{d[v]}{\sum_{w \in V_N} d[w]}.$$

The denominator in this fraction is nonzero since a path from v_0 to v_N exists.

For $i > 0$, to find v_{i-1} given v_i look at the vertices with edges going into v_i. Choose v_{i-1} from this set, selecting v with probability proportional to $d[v]$ times the probability of the edge from v to v_i. Thus the probability of selecting v where $v \xrightarrow[p]{*} v_i$, conditioning on v_i, will be

$$\frac{pd[v]}{\sum_{w \xrightarrow[q]{*} v_i} qd[w]}.$$

The denominator of this fraction is nonzero because a path from v_0 to v_N exists, and since (as can be seen inductivley) v_i belongs to one such path.

If we selected v_{i-1} such that $v_{i-1} \xrightarrow[p]{*} v_i$ then let $p_{i-1} = p$. Thus the probability of selecting v_{i-1} is $\frac{p_{i-1} d[v_{i-1}]}{\sum_{w \xrightarrow[q]{*} v_i} qd[w]}$, and the product of the probabilities of the edges of the selected path will be $\prod_{i=0}^{i<N} p_i$.

We now show that the probability of selecting path $v_0, \ldots, v_N$ is proportional to the product of the probabilities of the edges on this path. The probability of selecting the path is

$$\frac{d[v_N]}{\sum_{w \in V_N} d[w]} \prod_{i=1}^{N} \frac{p_{i-1} d[v_{i-1}]}{\sum_{w \xrightarrow[q]{*} v_i} qd[w]}.$$

But note that the denominators in the product are equal, by definition, to $d[v_i]$. So this expression becomes

$$\frac{d[v_N]}{\sum_{w \in V_N} d[w]} \prod_{i=1}^{N} \frac{p_{i-1} d[v_{i-1}]}{d[v_i]}.$$

Simplifying this telescoping product gives us

$$\frac{1}{\sum_{w \in V_N} d[w]} \prod_{i=0}^{i<N} p_i.$$

Note that $d[v_N]$ cancelled with a factor in the product, and $d[v_0] = 1$. Since the first part is a constant regardless of the chosen path, and the second is the product of the edge probabilities of the selected path, we have indeed sampled a path with probability proportional to its edge probabilities. Thus, we have also sampled a sequence according to our cover model, with the condition that it hides our message.

Assuming that we can sample from a set in linear time and space, this algorithm runs in linear time and space with respect to the size of the trellis graph. Due to the bound on this size, this is $O(N|C||\mathbb{B}|^h)$ time and space.

I now give an implementation that does not explicitly construct the graph. It notionally traverses the trellis graph, calculating d along the way, together with a dictionary *into*, where $into[v]$ represents the vertices that can reach v using one edge. Finally, it samples the path as described, starting from the end, and moving to the beginning.

In the following pseudocode, uninitialised keys of d correspond to 0, and uninitialised keys of *into* correspond to $\emptyset$. Moreover, the meaning of

$$\operatorname*{sample}_{x \in A} f(x)$$

is to sample some value $x \in A$, with probability $\frac{f(x)}{\sum_{y \in A} f(y)}$. The pseudocode is found in algorithm 1.

3.3.2 Maximal probability stegotext

Finding the maximal probability stegotext that hides the message is equivalent to finding the trellis graph path where the product of the edge probabilities is maximal. Note that we assume again that at least one stegotext exists that hides the message. By substituting edge probabilities with their negative logarithms, and noting that the trellis graph is directed and acyclic, we reduce the initial problem to the single-source shortest path problem in a directed acyclic graph. This can be solved in linear time and space [16, Chapter 24.2] with respect to the size of the trellis graph. Using the bound proven on the size of the trellis graph, this approach allows us to solve the problem in $O(N|C||\mathbb{B}|^h)$ time and space.

I now present a pseudocode implementation that does *not* explicitly construct the

Algorithm 1 Sampling stegotexts

```
1: procedure StegotextSampler(H, C, m)
2:     Initialise dictionaries d, into and sequence result.
3:     d[0][ϵ][⊥^w] = 1
4:     for i = 0 until N do
5:         for (mask, ctxt) in keys(d[i]) do
6:             for c in cand(ctxt) do
7:                 v = (i, mask, ctxt)
8:                 v' = nextVertex(v, c)
9:                 val = d[v]p(c | ctxt)
10:                if good(v') then
11:                    d[v'] = d[v'] + val
12:                    into[v'] = into[v'] ∪ {(val, v, c)}
13:    (mask, ctxt) = sample_{x∈keys(d[N])} d[N][x]
14:    v = (N, mask, ctxt)
15:    for i = N − 1 to 0 do
16:        (∗, v', c) = sample_{(x,∗,∗)∈into[v]} x
17:        result[i] = c
18:        v = v'.
19:    return result
```

trellis graph, in algorithm 2. This implementation will try to minimise the sum of the negative logarithms of the edge probabilities of the path that it searches for. We prefer this to maximising the product of the edge probabilities for two reasons: consistency with the reduction from the previous paragraph, and avoidoing floating point *underflow*. The algorithm is similar to the one in the previous section, only that now we use three dictionaries:

- *best*, where *best*[v] is the minimal sum of negative logarithms of edge probabilities for any path from v_0 to v.
- *prev*, where *prev*[v] is the penultimate vertex in the path found for *best*[v].
- *symb*, where *symb*[v] is the symbol on the edge from *prev*[v] to v.

Uninitialised keys of *best* correspond to ∞.

Algorithm 2 Finding maximal likelihood stegotext

```
1: procedure MaximalLikelihoodStegotext(H, C, m)
2:     Initialise dictionaries best, prev, symb, and sequence result
3:     best[0][ϵ][⊥^w] = 0
4:     for i = 0 until N do
5:         for (mask, ctxt) in keys(best[i]) do
6:             for c in cand(ctxt) do
7:                 v = (i, mask, ctxt)
8:                 v' = nextVertex(v, c)
9:                 cost = best[v'] − log p(c | ctxt)
10:                if good(v') ∧ best[v'] > cost then
11:                    best[v'] = cost
12:                    prev[v'] = v
13:                    symb[v'] = c
14:    (mask, ctxt) = arg min_{k∈keys(best[N])} best[N][k]
15:    v = (N, mask, ctxt)
16:    for i = N − 1 to 0 do
17:        result[i] = symb[v]
18:        v = prev[v]
19:    return result
```

Chapter 4

Applications

We now move on from the algorithms themselves to the ways in which they can be used. In this chapter we will explore three applications. The first shows us that our maximum likelihood stegotext algorithm is a generalisation of the steganographic uses of the Viterbi algorithm. The second builds on the first and pertains to image steganography. Given an image, we will modify it so that it hides our message, and an approximation of the UNIWARD distortion is minimised. The final application relates to natural language covers. We create a toy n-gram model, so that the cover size $|C|$ is polynomial with regards to the size of the corpus from which the model is generated. We then show how to apply the stegotext sampling algorithm to this model.

4.1 Links to the Viterbi algorithm

Consider the Viterbi algorithm as presented in [15]. The problem it solves is the following: given a cover of N bits[1], and a cost c_i for changing bit i, for a given syndrome trellis code with constraint length h, modify the cover so that it encodes a given message, with minimal cost. This problem can easily be solved using the maximal likelihood stegotext algorithm given earlier. The main difficulty is that our models use one probability distribution for all stegotext symbols regardless of position; whereas the problem now differentiates between symbols at different positions. To accommodate this we will encode positional information in Σ. This is only required to conform to the cover model scheme outlined earlier, and can

[1] While this section will assume that the message alphabet and the cover alphabet are equal to $\{0, 1\}$, the notions introduced can be easily generalised.

be handled implicitly in actual implementations – since the position is already included in each trellis graph vertex. Thus set $\Sigma = [0, N) \times \mathbb{B}, w = 1, enc(*, x) = x, \mathbb{C} = \Sigma \cup \{\perp\}$, and

$$p((i,x)|ctxt) = \begin{cases} \frac{e^{c'_{i,x}}}{\sum_{y \in \Sigma} e^{c'_{i,y}}}, ctxt = \perp \vee ctxt = (i+1, *) \\ 0, \text{ otherwise.} \end{cases}$$

Where $c'_{i,x}$ is 0 if the i^{th} bit of the cover is equal to x, and $-c_i$ otherwise. Note that under this model the negative log likelihood of a stegotext is equal to the cost of modifying the cover to match it, plus by a constant. Thus the maximal likelihood stegotext is also the minimal cost stegotext.

Using this model makes the maximal likelihood stegotext algorithm precisely match the Viterbi algorithm, allowing us to solve the problem in $O(N2^h)$ time and space. Although it seems at first that the trellis graph generated with this cover model would have $\Omega(N^2 2^h)$ size, it actually has $O(N2^h)$ size, since the vertices (other that v_0) have form $(i, mask, (i-1, c))$, where $i \in (0, N]$, $mask \in \{0,1\}^h$, $c \in \{0,1\}$, and each vertex has out degree at most 2.

By enlarging w, we can make p take into account consecutive sequences of bits when assigning costs, rather than just each bit individually. Thus, suppose that we have a locally nonlinear cost function $c : [0, N) \times \{0, 1, \perp\}^l \to \mathbb{R}$. Setting w to l and building p analogously to before, by applying the maximal likelihood stegotext algorithm we can find the stegotext s that minimises

$$\sum_{i=0}^{i<N} c\left(i, s\left(i-w, i\right]\right).$$

This variant has $O(N2^h 2^l)$ time and space complexity: each trellis graph vertex is of form $(i, ctxt, mask)$, and while $ctxt$ initially looks like it can be chosen in $(2N)^l$ ways due to including position information in symbols, in reality it can only be chosen in 2^l ways, since the positions are determined by i.

Thus the maximal likelihood stegotext algorithm can be seen as a locally nonlinear generalisation of the Viterbi algorithm. This way of using the algorithm will be used in the following section.

4.2 Image steganography applications

4.2.1 Problem setting

The application we initially considered for the maximal likelihood stegotext algorithm is in image steganography. We will use the form of the algorithm described in the previous section; thus, in order to apply the algorithm, we will create a locally nonlinear cost function for it to use. We start from the UNIWARD distortion function. In this setting, the covers are grayscale images. The images are H pixels tall and W pixels wide, and contain pixel values from 0 to 255. We mirror pad the images. We now define the UNIWARD distortion function.

Definition 13 (Uniward)**.** This definition will depend on constants $\sigma > 0$, $l \in \mathbb{N}$, $coef_{k,i,j} \in \mathbb{R}$, where $k = 1, 2, 3$ and $i, j \in [0, l)$, which for the purposes of this dissertation can be considered arbitrary. For an H by W image X define $f(k, i, j, X)$ by

$$f(k, i, j, X) = \sum_{i'=0}^{i'<l} \sum_{j'=0}^{j'<l} coef_{k,i',j'} X[i + i'][j + j'].$$

Thus $f(k, i, j, X)$ represents the sum of the elements of the convolution of $X[i, i+l)[j, j+l)$ with the matrix where the element at position (i, j) is $coef_{k,i,j}$.

Now, fixing two image X and Y define $g(k, i, j)$ to be

$$g(k, i, j) = \frac{|f(k, i, j, X) - f(k, i, j, Y)|}{\sigma + |f(k, i, j, X)|}.$$

Finally the UNIWARD distortion between X and Y is defined by

$$D(X, Y) = \sum_{k=1}^{3} \sum_{i=0}^{i<H} \sum_{j=0}^{j<W} g(k, i, j).$$

With this definition in mind, we state the problem. Given an image X, and a syndrome trellis code H, we must generate an image Y such that $D(X, Y)$ is minimised and Y hides our message m. To extract the message from Y, we iterate through Y in a fixed order, compute the remainders of each pixel value modulo 2, and multiply the resulting sequence with H. Thus, if $lin : [0, 256)^{H \times W} \to [0, 256)^{HW}$ is a function that orders the pixels of

its input in some way, and $mod_2(s) = \langle x \mod 2 : x \leftarrow s \rangle$, then $H \times mod_2(lin(Y))$ is the message that corresponds to Y.

4.2.2 Proposed solution

Now we show how to apply the algorithm. First, if a pixel has value x, then we will only modify it to x or $x+1$ if $x < 255$, or x and $x-1$ otherwise. This is needed in order to keep the size of Σ small: now, we can set $\Sigma = \{0, 1\}$, as this fully specifies the changes we allow ourselves to make. In other words, we will first generate stegotext $s \in \{0, 1\}^{HW}$, and then create Y by modifying the pixels of X as specified previously, so that $mod_2(lin(Y)) = s$. To create s, we specify a locally nonlinear cost function, and then apply the reduction from the previous section to find an s that minimises it. It is impossible to create a cost function that equals UNIWARD, while also keeping w small enough to be tractable, so we create an approximation.

Let $c : [0, H \times W) \times \{0, 1\}^w \to \mathbb{R}$ be the cost function according to which stegotext $s \in \{0, 1\}^{HW}$ is built. We set s so that

$$\sum_{i'=0}^{i<N} c\left(i', w\left(i' - w, i'\right]\right)$$

which is minimised by our algorithm is approximately equal to the UNIWARD distortion $D(X, Y)$. Note that w is arbitrary: making it larger increases the accuracy of our approximation, but also increases the running time.

The approach we use to make c is similar to the approach already used to create linear approximations of UNIWARD [15]: the distortion added by a set of local changes is approximated by the distortion it add if no other changes were done. More precisely, consider $c\,(i', ctxt)$. Let X' be equal X, but modified so that it matches $ctxt$ on $(i' - w, i']$ after linearisation, modulo 2. Set $c\,(i', ctxt)$ to $D(X, X')$. As it is inefficient to naively recalculate $D(X, X')$ for each possible value of i' and $ctxt$, we now describe an $O(HW2^w l^2)$ method to calculate these costs.

First, fix i'. Now, iterate through the 2^w ways of choosing $ctxt$. We attempt to maintain the current value of $D(X, X')$ throughout, together with the values of $f(k, i, j, X)$, $f(k, i, j, X')$ and $g(k, i, j)$ for all appropriate values of k, i, j. If we iterate through the

ways of choosing *ctxt* naively, each time *ctxt* changes, multiple bits of *ctxt* change at once. To avoid this, we choose *Gray code* order [7] instead. This is an ordering of the sequences in $\{0,1\}^w$ so that adjacent sequences differ by one bit. Therefore, after calculating the initial values of $f(k,i,j,X)$, $f(k,i,j,X')$, $g(k,i,j)$ and $D(X,X')$, in $O(HWl^2)$ time, as we iterate through the different ways of choosing *ctxt*, we must modify exactly one bit of X' for each such choice. Since Gray codes can be made cyclical, after finishing the iteration, we get back to where we began, so there is no need to recalculate the original values: they recalculate themselves!

How can we update these values after changing one bit of X'? First, consider the values of $f(k,i,j,X')$. Each pixel (i_0,j_0) in X' affects $O(l^2)$ values of $f(k,i,j,X')$ (those for which $i \in (i_0-l,i_0], j \in (j_0-l,j_0]$). Since each value of $f(k,i,j,X')$ is a linear combination of the values in X', when changing a pixel in X', we can change all of the values of $f(k,i,j,X')$ that it affects by adding some value. In particular, if pixel (i_0,j_0) of X' was previously set to y and is now set to y', when $i \in (i_0-l,i_0]$ and $j \in (j_0-l,j_0]$, the difference between the old and the new value of $f(k,i,j,X')$ is

$$coef_{k,i_0-i,j_0-j}(y'-y).$$

Each change of an $f(k,i,j,X')$ value affects one $g(k,i,j)$ value, and that value can be recalculated in constant time using the definition of g. The value of $D(X,X')$ is simply the sum of all of the $g(k,i,j)$ values, and thus can be immediately updated. So each one bit change of X' can be accommodated in $O(l^2)$ time.

Using this method allows us to compute all the required costs in $O(HWl^2 2^w)$ time. All that remains is to plug these costs into the maximum likelihood stegotext algorithm. This will then create the stegotext in $O(HW2^w 2^h)$ time, leading to an overall time complexity of $O(HW2^w(l^2+2^h))$ for this method.

While this was the first application we considered, when tried it turned out that it did not reduce the UNIWARD distortion significantly more than the linear approximation proposed previously in [15]. There are several reasons for this. Firstly, the linearisation step stops us from capturing many local dependencies. Second, the method through which we derived the nonlinear approximation may not be appropriate. Also, it turns out to be

more useful to increase h than w, and both h and w have similar effects on the time complexity. Nonetheless, we still believe that this algorithm opens up new avenues for distortion function creation.

We include, in table 4.1, the UNIWARD values given by this method for various values of h and w, averaged for the first 100 images in the BOSSBASE [12] image set.

Table 4.1: UNIWARD values

w \ h	5	6	7	8	9	10
1	80408.84	74045.42	69315.87	65483.08	62433.83	59784.44
2	79617.23	73285.35	68618.92	64822.16	61732.59	59112.73
3	79099.29	72770.75	68077.01	64269.46	61185.18	58581.96
4	78771.61	72419.74	67745.19	63933.44	60825.93	58216.83
5	78557.04	72180.95	67500.56	63677.95	60582.31	57964.37

4.3 Natural language models

These algorithms can also be applied to natural language. To do this, we build a toy *N-gram* model of natural language. Such models were introduced by Shannon [13]. We take advantage of model sparsity to make the approach efficient. Thus we not use any model smoothing (such as Laplace smoothing [3]), since smoothing would make the model less sparse. The model is not very sophisticated – it is used just to illustrate the power of the algorithms from before.

We will thus assume that our covers are drawn from a Markov chain of order w. The model is learned from a corpus: a set of texts, split into words and punctuation marks. To construct the model, set the probability of a symbol x appearing after a context c to be the number of times $c + x$ appears in the corpus, divided by the number of times $c + *$ appears in the corpus. [2]

Note that in this case the size of the model corresponds to the size of the corpus. This is the case since at most one element in a candidate set is added for each length $w + 1$ subsequence of the corpus. This leads to an interesting property: although, for arbitrary models the algorithms would run in exponential time with respect to w, for models based

[2] This makes the model exactly imitate the corpus for the first w symbols. To fix this, include in the distributions for contexts with $\perp$ information from all sentence beginnings.

on corpora, the complexity is polynomial with respect to the corpus size, regardless of w. The problem with raising w then becomes overfitting the model, rather than efficiency.

As for the encoding function, it is important to map words to $\mathbb{B}$ so that each value in $\mathbb{B}$ has an approximately equally sized preimage in Σ. This is necessary in order to be able to send as many messages as possible. This can be done approximately by approximating the probabilities, and applying dynamic programming – but it is intractable to find a perfect solution in general, as it is a generalisation of the PARTITION problem [10]. However, due to the density of such natural language models, this may be unnecessary – we can just arbitrarily assign each symbol of Σ its image in $\mathbb{B}$.

We have implemented this model in C++. The only significant optimisation is that symbols, contexts, masks and effects are represented by integers. A sample stegotext generated by this approach can be found in appendix A. The code is also included in appendix B. We also include tables 4.2, 4.3 that summarise some benchmarks run on the code. The corpus text is a prefix of the Iliad [17]. Each cell represents the average run time (in seconds) of the code, for a certain corpus length (in symbols), and message length (in bits). The constraint length of the syndrome trellis code was set to 7. Ten tests were run for each cell. Of the 8400 tests, eight failed (since the message happened to not be hidable). We created a linear model for the time, supposing that it depends on the product between the message length and the corpus size. This model had an R^2 of 84.7%, indicating that the algorithm shows the expected linear relationship between corpus size, message size, and run time.

Table 4.2: Sampler benchmarks

c. \ m.	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
26997	0.07	0.07	0.06	0.07	0.08	0.10	0.11	0.13	0.12	0.14	0.16	0.18	0.17	0.21	0.20	0.27
53994	0.16	0.19	0.23	0.27	0.26	0.38	0.56	0.68	0.92	1.02	1.04	1.48	1.66	1.81	2.25	2.28
80991	0.26	0.35	0.52	0.68	1.01	1.28	1.52	2.02	2.81	3.12	3.99	4.45	5.32	5.62	6.72	7.44
107988	0.56	0.79	1.14	1.57	2.51	3.18	4.11	5.33	6.49	7.77	8.63	10.25	11.99	12.54	13.67	15.68
134985	0.93	1.47	2.17	3.42	4.91	5.93	7.75	9.62	11.84	14.11	15.73	18.05	21.04	23.42	23.54	26.41
161982	1.52	2.23	3.58	5.05	7.03	9.51	12.07	15.09	17.46	19.54	24.51	27.08	30.36	33.75	35.75	32.66
188979	1.78	2.74	4.74	7.02	9.55	11.65	15.34	19.30	20.96	24.74	27.16	31.00	33.77	36.05	40.15	44.20
215976	2.42	4.05	6.09	9.05	13.56	16.75	20.53	25.23	28.19	32.33	36.66	43.02	48.59	53.94	57.29	61.41
242973	3.32	5.10	8.02	12.61	16.81	21.13	24.59	30.04	37.19	43.00	50.20	56.33	57.96	65.01	73.27	75.81
269970	3.96	6.67	10.13	14.41	19.53	25.46	31.56	37.62	45.71	51.16	55.34	61.77	66.27	77.61	78.92	90.33
296967	5.26	8.56	13.35	19.21	25.11	31.34	38.59	46.00	53.14	62.93	70.12	78.47	84.83	90.03	99.45	104.17
323964	6.00	9.79	15.21	21.75	30.06	37.74	47.27	56.34	66.10	74.39	79.06	89.91	99.73	103.11	115.63	104.72
350961	5.90	9.60	14.88	22.01	28.52	36.14	43.60	51.91	58.41	65.25	74.97	79.78	88.53	100.78	104.93	113.16
377958	7.38	12.06	18.72	27.99	36.14	47.59	57.01	67.22	83.90	87.57	95.19	109.56	116.80	131.68	138.23	145.55
404955	8.37	14.30	21.35	31.32	43.68	54.87	66.23	79.09	90.30	99.48	106.15	125.01	130.71	146.27	159.77	172.73
431952	8.73	14.23	22.24	32.25	42.23	50.97	61.95	75.07	82.65	97.17	106.65	115.56	124.06	133.27	149.21	159.82
458949	9.79	15.65	24.61	35.16	46.95	56.77	69.61	81.06	91.38	104.45	114.75	128.19	139.22	153.73	166.83	168.55
485946	11.68	20.86	31.12	44.52	58.93	72.76	87.87	102.17	110.63	128.90	145.57	159.25	177.37	199.51	208.84	219.91
512943	18.66	28.95	43.52	63.79	84.57	107.78	124.39	152.47	175.58	204.27	235.80	239.05	274.22	295.16	297.72	281.67
539940	18.99	30.69	46.34	67.90	90.23	114.34	135.23	149.13	178.56	225.84	250.65	270.83	293.58	323.01	297.03	291.15
566937	25.01	40.17	60.92	85.78	110.13	136.03	162.52	195.93	228.23	252.06	264.04	274.55	305.62	329.07	373.45	372.45
593934	25.28	43.75	65.52	90.84	123.74	143.69	183.16	206.45	230.28	256.98	281.79	326.35	360.05	384.83	420.23	468.30
620931	32.12	51.54	75.87	105.75	137.33	167.06	182.80	210.75	240.25	262.27	310.30	318.60	353.52	378.44	421.14	461.32
647928	31.79	50.76	81.08	107.98	140.44	170.68	193.86	236.79	253.00	305.94	338.83	343.83	407.06	469.06	479.54	484.34
674925	32.18	52.12	83.83	109.28	146.52	158.80	204.43	246.52	271.67	316.92	327.13	382.44	420.88	435.47	506.09	460.52
701922	31.69	62.24	93.22	122.08	162.59	189.96	241.13	255.27	309.56	313.44	351.64	435.17	414.38	485.30	550.78	612.73
728919	34.64	60.35	91.83	128.08	180.36	206.25	225.79	292.68	340.77	346.63	394.07	470.55	518.90	476.14	594.02	630.66
755916	30.62	63.75	89.10	130.00	161.77	202.86	245.09	279.04	304.01	352.11	389.55	440.49	478.07	485.84	550.99	667.98

Table 4.3: Sampler benchmarks

c. \ m.	24	25	26	27	28	29	30	31	32	33	34	35	36	37
26997	0.28	0.32	0.34	0.37	0.40	0.29	0.45	0.63	0.39	0.52	0.56	0.60	0.59	0.62
53994	2.78	3.33	3.40	3.82	3.96	4.31	4.70	4.64	5.22	5.58	6.08	6.35	5.94	6.85
80991	7.83	8.99	9.55	9.89	11.06	12.44	12.46	13.28	14.02	15.06	15.79	15.65	16.61	17.27
107988	16.71	18.21	20.02	21.28	22.50	23.96	25.61	27.50	28.09	28.93	32.21	32.33	34.10	36.93
134985	27.31	30.95	33.19	36.84	37.95	39.80	40.92	44.59	43.07	42.28	45.01	50.56	56.11	52.92
161982	38.59	44.77	48.88	46.71	45.14	47.97	49.02	59.96	60.91	60.62	62.59	66.03	68.05	72.22
188979	47.83	50.25	53.21	59.03	61.46	67.38	67.41	71.35	75.07	77.64	77.88	83.83	90.39	89.88
215976	61.49	72.76	72.53	78.09	82.23	86.28	91.99	91.89	97.13	103.11	112.22	107.25	110.78	119.56
242973	79.32	82.98	92.90	100.23	102.79	109.45	109.17	113.54	119.37	129.90	132.89	143.96	150.01	155.50
269970	97.35	102.44	107.76	113.69	127.48	126.01	133.72	138.56	140.17	149.71	168.41	171.54	175.02	183.32
296967	113.35	117.58	122.73	127.70	137.94	144.71	166.74	164.73	167.59	177.53	174.59	183.09	194.91	202.38
323964	106.74	114.26	115.16	126.79	134.33	140.96	145.24	156.85	161.55	164.74	170.61	185.68	188.97	201.76
350961	119.86	124.45	132.77	144.91	152.32	156.50	171.63	178.26	179.61	202.34	276.46	244.29	275.75	289.83
377958	155.17	164.72	175.56	183.19	194.22	206.19	209.46	219.48	238.06	240.64	258.90	272.84	294.58	293.07
404955	177.36	198.49	196.19	210.56	206.96	205.40	209.64	222.70	226.52	235.69	245.41	252.21	271.66	273.48
431952	166.79	179.44	192.27	201.30	214.33	217.41	224.87	247.92	252.11	257.80	268.99	282.16	296.94	302.50
458949	183.96	197.36	213.48	225.45	254.06	265.60	282.29	326.80	301.29	323.78	340.69	353.28	442.63	385.86
485946	239.84	243.70	271.15	294.99	287.05	336.97	340.38	343.14	454.10	529.62	488.53	507.02	542.09	581.26
512943	337.34	386.02	339.60	432.43	467.35	387.63	436.09	461.48	517.66	522.24	512.32	582.67	639.29	630.99
539940	335.75	394.32	416.51	416.72	388.29	395.19	370.41	372.80	439.21	492.47	514.10	553.17	655.21	818.38
566937	402.21	414.46	452.94	506.19	509.23	508.12	541.00	560.21	579.06	596.75	630.34	647.55	706.18	725.74
593934	482.15	509.45	535.40	572.71	599.99	604.35	583.39	605.70	629.76	672.87	711.17	778.72	762.11	840.88
620931	474.89	485.45	567.62	591.87	630.30	682.28	722.79	786.24	782.34	880.56	870.35	870.14	927.52	974.15
647928	584.53	612.04	638.71	809.24	811.05	815.36	825.69	902.56	967.37	983.35	984.70	1036.24	1014.01	901.65
674925	606.99	641.42	698.38	671.14	851.67	859.27	884.38	926.26	910.71	1068.75	1066.54	1070.89	1145.31	1124.43
701922	670.83	683.84	787.24	822.90	892.70	892.83	1005.30	983.64	1081.82	1024.57	1081.66	1133.95	1098.36	1151.69
728919	638.26	673.09	860.12	767.14	903.28	936.40	1050.01	1116.92	1146.78	1146.79	1198.83	1234.64	1193.45	1160.38
755916	689.84	807.60	884.57	944.65	984.60	1231.99	1359.92	1311.22	1308.71	1354.06	1431.17	1384.27	1344.67	1275.62

Chapter 5

Final considerations

5.1 Context in the field

We believe that the algorithms shown in this dissertation successfully generalise and unify several different directions that have appeared so far in steganography. Moreover, they offer a novel method of sampling in steganography – as far as we are aware, this is the first instance in which syndrome trellis codes have been used for sampling – and an alternative to the Gibbs construction in steganography [6] for using nonlinear distortion functions.

The Gibbs construction is a technique that can be used to find the minimal cost stegotext given a nonlinear cost function. It partitions the stegotext into disjoint parts, so that, conditioning on all the other parts, the cost of a part is linear. Then, it splits the message into submessages, one for each part. It then repeatedly traverses the stegotext until the cost is low enough. In a traversal, it iterates through the parts, and embeds the required submessage in the part, conditioning on the current value of the other parts, using the Viterbi algorithm. It can be proved that the stegotext tends to the minimal cost one under certain conditions.

Our maximal likelihood algorithm has several advantages when compared to the Gibbs construction. First, it is not always clear when to stop the Gibbs construction (i.e. how many traversals to perform) – whereas, for our algorithm, the time the algorithm still needs until it finishes is always known. Secondly, it is not always trivial to decide how much information to hide in each part in the Gibbs construction (one part may have higher capacity than the other); no such complications appear in our algorithm. Also, when the

stegotext alphabet is very large, and the cover distribution is relatively sparse (such as in natural language), the Gibbs construction may stall. For example, suppose we use a trigram model of text, splitting our stegotext into three parts (containing positions of form $3k$, $3k+1$, $3k+2$). Imagine if the initial stegotext is "*green mirror camels but now*". When we embed in the first part, the likelihood for the fourth word is determined by the previous two words, which we consider fixed: "*mirror camels*". But now, all possible words are very unlikely, and thus any improvement in total likelihood will be small. We see that the Gibbs construction is sensitive to the choice of initial stegotext, and in such models it may be difficult to find a good one. On the other hand, the Gibbs construction also has advantages: it easily handles two-dimensional cost nonlinearities, and it also handles both forward and backwards dependencies.

5.2 Limitations

The the methods shown here have three main limitations. First, they take the model and the syndrome trellis code as a given, and will only work if a stegotext that hides the message exists. The syndrome trellis code cannot be modified post-hoc to fix this (since such modifications would also need to be communicated); however, the model can be. It is usually possible to carefully design the *enc* function so that, given any context, of that context's candidate symbols, at least one exists that hides each element of $\mathbb{B}$. Of course *enc* would then need to know the context, but this is easy to arrange.

Another limitation is that the nonlinear dependencies that are modelled are inherently one dimensional. This leads to problems when trying to hide in images. When using additive costs we can randomly order the pixels, however now, in order to preserve the locality of image dependencies, we must go through the image in some continuous path. Moreover, not all two dimensional dependencies can be captured in one dimensional dependencies, since this path will not necessarily visit nearby pixels in close succession.

Finally, it is not necessarily the case that, when message are randomly selected, the distribution of the outputs of our stegotext sampling algorithm matches the cover distribution. Our algorithm produces stegotexts that have the same distribution as the cover *when conditioning on the message*; however, what is actually observed by the warden is

the unconditional distribution.

5.3 Further Work

Several directions remain to be explored further. One is to determine, using natural language processing or machine learning, the empirical detectability of these algorithms using various cover models and settings. In the context of natural language watermarking using an N-gram model and the stegotext sampling algorithm, we do not expect very good results: the model is just a toy example, and a warden that has a more sophisticated linguistic model should be able to detect stegotext built with this approach.

Another interesting direction is finding the detectability of images sampled using probability distributions derived from traditional steganographic distortion functions such as UNIWARD or WOW. Thus, rather than find the image that minimises these distortion functions, we sample from a distribution built around these minima. This technique may make the resulting images less detectable, since it hides the fact that a specific cost function was used.

A third avenue is to devise cover models that take into account the strengths and limitations of these algorithms. For image steganography this means a cost function that works around the inherent one-dimensionality of our algorithms, while also taking advantage of the fact that the stegotext does not need to be partitioned into conditionally independent parts (as with Gibbs sampling). For natural language watermarking, this means creating more sophisticated models, that maintain a degree of sparsity.

Finally, as was noted in the previous section, the sampling algorithm doesn't sample from the unconditional distribution of covers. We believe that it might be possible to sample the stegotext in such a way that – even though, for any fixed message, the stegotext distribution conditioning on that message may be different from the cover distribution conditioning on that message – the unconditional distribution of the stegotext perfectly matches the distribution of the covers.

5.4 Reflections

I am grateful to have had the opportunity to do this project, since it has given me the chance to learn more about a less commonly thought-about part of computer science. The first steps I took were implementing the standard Viterbi algorithm. After discussing about the Gibbs construction in steganography with my project supervisor, I had the idea to extend the Viterbi algorithm to locally nonlinear cost functions (which later became the maximal likelihood stegotext algorithm). After testing the algorithm on the UNIWARD cost function, and unsuccesfully trying to devise new cost functions, we realised that the algorithm would be well ilustrated by a textual N-gram model. This was because of the tension between the inherently linear way that the algorithm works, and the two dimensional nature of images: texts are also one dimensional, so they work well with the algorithm.

After implementing the maximal likelihood algorithm on this model, and noting that it often got stuck in cyclical, agrammatical constructions – probably due to the fact that the message constraint was not very restrictive, and the most likely path could be fairly free, and thus would repeat a very likely sequence of words. Trying to fix this, I first thought about fixing some of the words a-priori (this doesn't work well because the algorithm has no look-ahead, so it can't plan for the future fixed words). Then, I though about somehow randomising the word selected next (not yet sampling a stegotext, but finding the maximally likely stegotext, where the candidate function is randomly restricted). After thinking about this for a while, I realised that it was horribly inelegant: why not just randomise everything? This lead to the stegotext sampling algorithm. The way I reached this algorithm taught me something about the research process: when exploring an idea, rather than forcing it to work when it seems like it isn't the best fit, it is better to dig deeper and find elegant applications and ideas.

Appendix A

Example stegotexts

Figure A.1 contains an example stegotext. This is the raw output of the code in in the following appendix. When transmitted, it can be formatted properly, as this does not modify the hidden message.

these will give hector son of priam , even in single combat . thus did they converse . meanwhile thetis came to the house of hades . would that he had sooner perished he will restore , and will add much treasure by way of amends . go , therefore , into battle , and show yourself the man you have been always proud to be . idomeneus answered , i will tell the captains of the ships and all the achaeans with them ere i went back to ilius . but why commune with myself in this way ? can you not see that the trojans are within the wall some of them stand aloof in full armour , we shall have knowledge of him in good earnest . glad indeed will he be who can escape and get back to ilius , but darkness came on too soon . it was this alone that saved them and their ships upon the seashore . now , therefore , that you have been promising her to give glory to hector . meanwhile the rest of the gifts , and laid them in the ship s hold ; they slackened the forestays , lowered the mast into its place , and rowed the ship to the place where his horses stood waiting for him at the rear of our ships , and let it have well - made gates that there may be a way through them for their chariots , and close outside it they dug a trench deep and wide all round it , and as the winning horse in a chariot race strains every nerve when he is flying over the plain , killing the men and bringing in their armour , though they were still lads and unused to fighting . now there is a high mound before the city , rising by itself upon the plain . and now iris , fleet as the wind , from the heights of ida to the lofty summits of olympus . she went to priam s house , and found weeping and lamentation therein . his sons were seated round their father in the outer courtyard , and their raiment was wet with tears as she prayed that they would kill her son and erinys that walks in darkness and knows no ruth heard her from erebus . then was heard the din of battle about the gates of calydon , and the dull thump of the battering against their walls . thereon the elders of the aetolians besought meleager ; they sent the chiefest of their priests , and begged him to come out and help them , promising him a great reward . they bade him choose fifty plough - gates , the most fertile in the plain of calydon , the one - half vineyard and the other open plough - land . the old warrior oeneus implored him , standing at the threshold of his

Figure A.1: Example stegotext

Appendix B

Text sampler code

I now include the code for the text sampler. It is written in C++. I first show the `makefile` used, then the header files, then the source code files.

makefile

```
SRCDIR := src
OBJDIR := obj
DEPDIR := .deps

TARGET := sampler

SRCS := $(wildcard $(SRCDIR)/*.cxx)
OBJS := $(patsubst $(SRCDIR)/%.cxx,$(OBJDIR)/%.o,$(SRCS))
DEPS := $(patsubst $(SRCDIR)/%.cxx,$(DEPDIR)/%.d,$(SRCS))

DEPFLAGS = -MT $@ -MMD -MP -MF $(DEPDIR)/$*.d
CPPFLAGS += -std=c++11 -O3 -g

sampler : $(OBJDIR) $(OBJS)
        $(CXX) $(CPPFLAGS) $(OBJS) -o $(TARGET)

clean :
        -rm -rf $(OBJDIR) $(DEPDIR) $(TARGET)

$(OBJDIR)/%.o : $(SRCDIR)/%.cxx $(OBJDIR) $(DEPDIR)/%.d | $(DEPDIR)
        $(CXX) $(DEPFLAGS) $(CPPFLAGS) -c $< -o $@

$(DEPDIR): ; @mkdir -p $@

$(OBJDIR): ; @mkdir -p $@

$(DEPS):

include $(wildcard $(DEPS))
```

src/model.hxx

```
#ifndef MODEL_H
#define MODEL_H

#include <vector>
#include <map>
#include <string>
using namespace std;

// Each symbol will be represented by an integer code.
using symbol = unsigned;

// A priori I fix that _|_ is represented by 0.
constexpr unsigned bottom_symbol = 0;
```

```
// Each possible context will be represented by an integer code.
using context = unsigned;

// Likewise, I fix that the context containing only _|_ is represented by
↪  0.
constexpr unsigned bottom_context = 0;

class model {
    unsigned clen;
    map<string, symbol> symbol_to_code;
    vector<string> code_to_symbol;

    map<vector<symbol>, context> sequence_to_context;
    vector<map<symbol, pair<float, context>>> following_symbols;
    vector<float> total_count;

    // Tokenises a string, splitting it into words and punctuation,
    // and adding in symbol meanings to model.
    vector<symbol> tokenise(string);
public:
    // Empty model constructor.
    model(unsigned context_length);

    // Create a model from an input text.
    static model model_from_text(unsigned context_len, string);

    // Translate a symbol code back into its meaning.
    string symbol_meaning(symbol);
    // Translate a concrete symbol into its code. Adds symbol
    // to model if not yet encountered.
    symbol symbol_name(string);

    // Translate a sequence of symbols into its context code.
    // Adds context to model if not yet encountered.
    context context_name(const vector<symbol>&);

    // Adds one to the count of a certain context/symbol pair.
    void increment_model(const vector<symbol>&, symbol);

    // Given a context code, return possible following symbols,
    // together with relevant probabilities. Also include the
    // contexts that would result.
    const map<symbol, pair<float, context>>& cand_and_p(context) const;

    // Given a symbol, encode it.
    unsigned encode(symbol) const;

    // Given a string of symbols, encode the symbols.
    vector<unsigned> encode_sequence(vector<symbol>);

    // Find number of contexts.
    unsigned context_count() const;
};

#endif
```

src/sampler.hxx

```
#ifndef SAMPLER_H
#define SAMPLER_H

#include <vector>
#include <random>
#include "trellis.hxx"
#include "model.hxx"
using namespace std;

// Sample according to c, conditioning that h.recover(c.encode(return
↪  value)) is m.
vector<symbol> conditional_sample(const model& c, const trellis& h,
↪  vector<unsigned> m, unsigned seed);
```

```
#endif
```

src/trellis.hxx

```
#ifndef TRELLIS_H
#define TRELLIS_H

#include <vector>
#include <random>
using namespace std;

using mask = unsigned;

class trellis{
    int height, mlen, slen;
    vector<int> first, last, col;
public:
    // Generate trellis of a given height and sizes, from a seed.
    trellis(int h, int ml, int sl, int seed);

    // Get trellis height.
    int h() const;

    // Get message length.
    int message_len() const;

    // Get stegotext length.
    int stego_len() const;

    // For stegotext position i, gets first position
    // affected by i. By convention, fst(-1) = 0,
    // fst(stego_len()) = message_len()
    int fst(int) const;

    // For stegotext position i, gets first position
    // not affected by i. By convention, fst(-1) = 0,
    // fst(stego_len()) = message_len()
    int lst(int) const;

    // dFst(x) = fst(x) - fst(x - 1)
    int dFst(int) const;

    // dLst(x) = lst(x) - lst(x - 1)
    int dLst(int) const;

    // len(x) = lst(x) - fst(x)
    int len(int) const;

    // Effect of stegosymbol x on message, as a bitmask,
    // from left to right, in big-endian order.
    int effect(int) const;

    // Apply trellis matrix to stegosequence.
    vector<unsigned> recover(vector<unsigned>) const;
};

#endif
```

src/main.cxx

```
#include <fstream>
#include <iostream>
#include "trellis.hxx"
#include "model.hxx"
#include "sampler.hxx"
using namespace std;

static constexpr int message_len = 50;
static constexpr int stego_len = 500;
static constexpr int trellis_height = 7;

int main(){
    int seed = time(nullptr);
```

```
    // I happened to use
    ifstream model_input("illiad.txt");

    string model_contents{
        istreambuf_iterator<char>(model_input),
        istreambuf_iterator<char>()};

    model m = model::model_from_text(4, model_contents);
    trellis t(trellis_height, message_len, stego_len, seed);

    vector<unsigned> message(message_len);
    random_device rd;
    for(auto& x : message) x = uniform_int_distribution<int>(0, 1)(rd);

    auto ret = conditional_sample(m, t, message, seed);

    auto verif = t.recover(m.encode_sequence(ret));

    for(auto x : ret)
        cout << m.symbol_meaning(x) << ' ';
    cout << endl;

    for(auto x : message)
        cerr << x << ' ';
    cerr << endl;

    for(auto x : verif)
        cerr << x << ' ';
    cerr << endl;

    return 0;
}
```

src/model.cxx

```
#include "model.hxx"
#include <iostream>
#include <fstream>
using namespace std;

vector<symbol> model::tokenise(string s){
    // Return value.
    vector<symbol> ret;

    // The current word in the following iteration.
    string current;

    // Add in the current to the result, if appropriate, and empty it.
    auto add_and_empty = [&](){
        if(!current.empty()){
            ret.push_back(symbol_name(current));
            current.clear();
        }
    };

    for(auto c : s){
        if(isalnum(c))
            current.push_back(tolower(c));
        else
            add_and_empty();
        if(ispunct(c))
            ret.push_back(symbol_name(string(1, c)));
    }
    ofstream g("token-log.txt");
    for(auto x : ret)
        g << symbol_meaning(x) << '\n';
    return ret;
}

// Initialise the model with a-priori information about bottom symbol
// and bottom context.
model::model(unsigned context_len):
```

```
    clen(context_len),
    symbol_to_code({ { string{""}, bottom_symbol } }),
    code_to_symbol({ string{""} }),
    sequence_to_context({ { vector<symbol>(context_len, bottom_symbol),
    ↪  bottom_context } }),
    following_symbols({ { { bottom_symbol, { 0.0, bottom_context } } }
    ↪  }),
    total_count({ 1.0 }){}

model model::model_from_text(unsigned context_len, string s){
    // Return value.
    model ret(context_len);

    // First tokenise model, adding in all relevant symbols:
    auto toks = ret.tokenise(s);

    // Now for all context/symbol pairs, just increment model.

    // This variable will hold the context throughout.
    vector<symbol> ctx(context_len, bottom_symbol);

    // Fills ctx with contents ending before symbol j, assuming that
    // symbol i is the first symbol in the text.
    auto fill_context = [&](unsigned i, unsigned j){
        unsigned amount = min(j - i, context_len);
        fill(begin(ctx), begin(ctx) + context_len - amount,
        ↪  bottom_symbol);
        copy(begin(toks) + j - amount, begin(toks) + j, begin(ctx) +
        ↪  context_len - amount);
    };

    cerr << "Making model" << endl;

    for(unsigned i = 0; i < toks.size(); ++i){
        if(i > 0) cerr << "\r";
        cerr << "At model making step " << i;

        fill_context(0, i);
        ret.increment_model(ctx, toks[i]);
    }

    // Now add in special logic for "sentence beginnings". This
    // makes the start of our modelled covers more diverse.
    for(unsigned i = 0; i < toks.size(); ++i){
        cerr << "\rAt model making step " << toks.size() + i;

        if(ret.symbol_meaning(toks[i]) != ".")
            continue;

        for(unsigned j = i + 1; j < i + context_len + 1 && j <
        ↪  toks.size(); ++j){
            fill_context(i + 1, j);
            ret.increment_model(ctx, toks[j]);
        }
    }
    cerr << endl;


    return ret;
}

string model::symbol_meaning(symbol s){
    return code_to_symbol[s];
}

symbol model::symbol_name(string s){
    auto it = symbol_to_code.find(s);
    if(it == end(symbol_to_code)){
        code_to_symbol.push_back(s);
```

```
        symbol_to_code.emplace_hint(it, s, symbol_to_code.size());
        return symbol_to_code.size() - 1;
    }
    return it->second;
}

context model::context_name(const vector<symbol>& v){
    auto it = sequence_to_context.find(v);
    if(it == end(sequence_to_context)){
        following_symbols.emplace_back();
        total_count.push_back(0);
        sequence_to_context.emplace_hint(it, v,
        ↪  sequence_to_context.size());
        return sequence_to_context.size() - 1;
    }
    return it->second;
}

void model::increment_model(const vector<symbol>& v, symbol s){
    auto c1 = context_name(v);

    // Get outgoing edges from map.
    auto it = following_symbols[c1].find(s);

    // If this is a new transition.
    if(it == end(following_symbols[c1])){
        // Construct next context.
        vector<symbol> target(begin(v) + 1, end(v));
        target.push_back(s);

        // Add transition
        following_symbols[c1][s] = make_pair(1.0, context_name(target));
    }
    // Otherwise just increment model directly.
    else
        it->second.first += 1.0;

    // And increment total count.
    total_count[c1] += 1.0;
}

const map<symbol, pair<float, context>>& model::cand_and_p(context c)
↪  const{
    return following_symbols[c];
}

unsigned model::encode(symbol s) const{
    return __builtin_popcount(s) % 2;
}

vector<unsigned> model::encode_sequence(vector<symbol> v){
    vector<unsigned> ret;

    for(auto x : v)
        ret.push_back(encode(x));
    return ret;
}

unsigned model::context_count() const{
    return sequence_to_context.size();
}
```

src/sampler.cxx

```
#include "sampler.hxx"
#include <iostream>
#include <cassert>
using namespace std;

// Node in trellis tree. Rather than explicitly maintain all back edges
// in the graph, I will choose a back edge randomly as I go along, thus
```

```
// needing constant space per node. Interestingly this reduces the
// trellis graph to a tree.
struct node{
    node *father = nullptr;
    float d = 0;
    symbol father_sym = bottom_symbol;
    int position;

    node(int pos):
        position(pos){}
};

vector<symbol> conditional_sample(const model& c, const trellis& h,
↪   vector<unsigned> m, unsigned seed){
    cerr << "Doing conditional sample" << endl;
    minstd_rand mt(seed);

    const unsigned hh = h.h(), hmask = (1 << hh) - 1;

    // Firstly, this is useful for checking if a state is good:
    vector<unsigned> good_check_vec(h.stego_len());
    for(int i = 0; i < h.stego_len(); ++i)
        for(int j = h.fst(i); j < h.fst(i + 1); ++j)
            good_check_vec[i] = 2 * good_check_vec[i] + m[j];

    // The root of the trellis graph.
    node *root = new node { 0 };

    // The current layer is indexed by (mask, context) pairs. But
    ↪  holding
    // these as pairs in the inner loop is innefficient. Thus I will
    ↪  index
    // (mask, context) as mask | context << hh
    //
    // The root ought properly to be only at key 0, but due to the fact
    ↪  that
    // these entries are accessed only if mentioned in visit_now or
    ↪  visit_next,
    // and entries at wrong positions in the text are counted as being
    ↪  invalid,
    // I can use root as a dummy value, to remove a nullptr check later.
    vector<node*> current_layer{c.context_count() << h.h(), root },
        prev_layer{c.context_count() << h.h(), root };

    // Nodes that we visit on next layer.
    vector<unsigned> visit_now, visit_next;

    // We visit the trellis root, which is properly at key 0.
    visit_next.push_back(0);

    // Advance h.stego_len() the current layer.
    for(int i = 0; i < h.stego_len(); ++i){
        const auto mask_lim = (1 << h.len(i)) - 1;
        const auto my_dLst = h.dLst(i);
        const auto my_effect = h.effect(i);

        if(i > 0)
            cerr << '\r';
        cerr << "At conditional sample step " << i;

        swap(current_layer, prev_layer);
        swap(visit_now, visit_next);
        visit_next.clear();

        for(auto current_idx : visit_now){
            const auto current_node = prev_layer[current_idx];
            const auto msk = current_idx & hmask;
            const auto ctx = current_idx >> hh;
            const auto d = current_node->d;
```

```
            for(const auto& t : c.cand_and_p(ctx)){
                const auto b = c.encode(t.first);
                const auto msk_ = ((msk << my_dLst) ^ (b * my_effect)) &
                ↪  mask_lim;

                if(good_check_vec[i] ^ (msk_ >> (h.lst(i) - h.fst(i +
                ↪  1)))) continue;

                const auto ctx_ = t.second.second;
                const auto p = t.second.first;
                const auto k = msk_ | ctx_ << hh;

                // No nullptr check since current_layer never contains a
                ↪  null pointer.
                if(current_layer[k]->position != i + 1){
                    current_layer[k] = new node { i + 1 };
                    visit_next.push_back(k);
                }
                const auto next_node = current_layer[k];

                if(!bernoulli_distribution(next_node->d / (next_node->d +
                ↪  d * p))(mt)){
                    next_node->father = current_node;
                    next_node->father_sym = t.first;
                }
                next_node->d += d * p;
            }
        }
    }
    cerr << endl;

    // Select a final node from the current (i.e. last) layer, using
    // a similar strategy to before.
    float d = 0;
    node* me = nullptr;

    for(auto x : visit_next){
        auto other = current_layer[x];
        if(!bernoulli_distribution(d / other->d)(mt))
            me = other;
        d += other->d;
    }

    // Now backwalk through the graph to reconstitute the result.
    vector<symbol> ret(h.stego_len());

    for(int i = 0; i < h.stego_len(); ++i){
        ret.rbegin()[i] =  me->father_sym;
        me = me->father;
    }

    return ret;
}
```

src/trellis.cxx

```
#include "trellis.hxx"
#include <iostream>
using namespace std;

trellis::trellis(int h, int ml, int sl, int seed):
    height(h),
    mlen(ml),
    slen(sl),
    first(sl),
    last(sl),
    col(sl)
{
    mt19937 mt(seed);

    // Width of a trellis block.
```

```
    int width = (sl - h + ml - 1) / ml;

    for(int i = 0; i < slen; ++i){
        first[i] = i / width;
        last[i] = min(ml, first[i] + height);
        col[i] = uniform_int_distribution<int>(0, 1 << (last[i] -
        ↪  first[i]))(mt);
    }
};

int trellis::h() const{
    return height;
}

int trellis::message_len() const{
    return mlen;
}

int trellis::stego_len() const{
    return slen;
}

int trellis::fst(int i) const{
    return i < 0 ? 0 : i >= slen ? mlen : first[i];
}

int trellis::lst(int i) const{
    return i < 0 ? 0 : i >= slen ? mlen : last[i];
}

int trellis::dFst(int i) const{
    return fst(i) - fst(i - 1);
}

int trellis::dLst(int i) const{
    return lst(i) - lst(i - 1);
}

int trellis::effect(int x) const{
    return col[x];
}

int trellis::len(int x) const{
    return lst(x) - fst(x);
}

vector<unsigned> trellis::recover(vector<unsigned> s) const{
    vector<unsigned> m(message_len());
    for(int i = 0; i < stego_len(); ++i){
        if(s[i] == 0) continue;
        for(int j = fst(i); j < lst(i); ++j)
            m[j] ^= (effect(i) >> (lst(i) - j - 1)) & 1;
    }
    return m;
}
```

Bibliography

[1] R. Anderson and F. Petitcolas. On the limits of steganography. *IEEE Journal on Selected Areas in Communications*, 16:474–481, 12 1998.

[2] P. Bas. Natural steganography: cover-source switching for better steganography, 07 2016.

[3] H. Schütze C. D. Manning, P. Raghavan. *Introduction to Information Retrieval*. Cambridge University Press, USA, 2008.

[4] E W. Dijkstra. Why numbering should start at zero, August 1982 (accessed May 7, 2020). http://www.cs.utexas.edu/users/EWD/ewd08xx/EWD831.PDF.

[5] H. van Tilborg E. Berlekamp, R. McEliece. On the inherent intractability of certain coding problems (corresp.). *IEEE Transactions on Information Theory*, 24(3):384–386, 1978.

[6] T. Filler and J. Fridrich. Gibbs construction in steganography. *Information Forensics and Security, IEEE Transactions on*, 5:705 – 720, 01 2011.

[7] F. Gray. Pulse code communication, 1947.

[8] V. Holub and J. Fridrich. Designing steganographic distortion using directional filters. *WIFS 2012 - Proceedings of the 2012 IEEE International Workshop on Information Forensics and Security*, 12 2012.

[9] N. Hopper, L. von Ahn, and J. Langford. Provably secure steganography. *IEEE Transactions on Computers*, 58(5):662–676, May 2009.

[10] R. M. Karp. Reducibility among combinatorial problems. *Complexity of Computer Computations*, 40, 01 1972.

[11] J. Nakamura. *Image Sensors and Signal Processing for Digital Still Cameras*. Optical Science and Engineering. CRC Press, 2017.

[12] T. Filler. B P. Bas, T. Pevný. BossBase, March 2011. http://webdav.agents.fel.cvut.cz/data/projects/ stegodata/BossBase-1.01-cover.tar.bz2.

[13] C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27(3):379–423, 1948.

[14] G. J. Simmons. The prisoners' problem and the subliminal channel. In David Chaum, editor, *CRYPTO*, pages 51–67. Plenum Press, New York, 1983.

[15] J. Judas T. Filler and J. Fridrich. Minimizing embedding impact in steganography using trellis-coded quantization. *IEEE Trans Inf Secur Forensics*, 6:754105, 02 2010.

[16] R. L. Rivest T. H. Cormen, C. E. Leiserson and C. Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

[17] Homer (trans. S. Butler). The Iliad, 1898 (accessed May 9, 2020). https://www.gutenberg.org/files/2199/2199-0.txt.

[18] J. Fridrich V. Holub and T. Denemark. Universal distortion function for steganography in an arbitrary domain. *EURASIP Journal on Information Security*, 1, 12 2014.

[19] A. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Trans. Inf. Theor.*, 13(2):260–269, September 2006.