TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik
Institut für Architektur verteilter Systeme
Lehrstuhl Datenschutz und Datensicherheit

DIPLOMARBEIT

Einbettung mit Trelliskodes

bearbeitet von

Benjamin Kellermann

geboren am 29. März 1980 in Dresden

zum
Erlangen des akademischen Grades

Diplominformatiker
(Dipl.-Inf.)

Betreuer:	Dr.-Ing. Dagmar Schönfeld
Verantwortlicher Hochschullehrer:	Prof. Dr. rer. nat. Andreas Pfitzmann
Tag der Einreichung:	14. Dezember 2007

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Steganographie ist die Kunst der versteckten Kommunikation. Anders als bei der Kryptographie, wo Nachrichteninhalte geschützt werden, kommt es in der Steganographie darauf an, die Existenz einer Nachricht zu verbergen. Nicht nur für Geheimdienste ist so etwas wichtig, auch kann damit gezeigt werden, dass Kryptographieverbote, wie sie heute noch in zahlreichen Ländern bestehen wirkungslos sind, da hier steganographisch versteckt kommuniziert werden kann.

Schon in der Antike wurde Steganographie praktiziert. Sklaven wurden Nachrichten auf den Kopf tätowiert. Nachdem die Haare lang genug waren konnte man keine Nachricht mehr erkennen. Im zweiten Weltkrieg wurden in Satzzeichen oder im i-Punkt Nachrichten versteckt, die man nur mit einem Mikroskop auslesen konnte, so genannte Mikropunkte. Seit der Digitalisierung der Steganographie versucht man Nachrichten in Bildern, Audiodateien oder anderen digitalen Medien zu verstecken.

Dabei wird versucht, Änderungen in der niederwertigsten Bitebene eines digitalen Mediums zu verstecken, in der Hoffnung, diese seien genügend zufällig, dass deren Änderung nicht auffällt.

Um nachzuweisen, ob bestimmte Daten zufällig sind oder nicht, existieren viele statistische Tests. Um einem Angriff eines solchen Tests auf ein Steganographiesystem zu entgehen, versucht man die Zahl der Änderungen in einem Medium zu minimieren. Hier entsteht ein typischer Konflikt zwischen der Menge der Daten, die eingebettet werden können und der Anzahl der Änderungen, die man in einem Medium vornehmen muss. 1998 milderte Ron Crandall diesen Konflikt etwas, indem er die Kanalkodes in die Steganographie einführte.[1] Er zeigte auf, wie man mit Hammingkodes mit maximal einer Änderung in $2^k - 1$ Bits k Bits einbetten kann. Eine ähnliche Einbettungsfunktion zeigte er mit dem perfekten Golaykode.

Um die Güte einer Einbettungsfunktion zu bewerten hat sich die Einbettungseffizienz etabliert. Diese bezeichnet, wie viele Änderungen in einem Medium pro eingebettetes Bit vorgenommen werden müssen.

[1] Ron Crandall, Some Notes on steganography. Posted on Steganography Mailing List, Dezember 1998 ⟨URL: `http://os.inf.tu-dresden.de/~westfeld/crandall.pdf`⟩.

Nachdem Crandall perfekte Kodes in die Steganographie einführte, gab es viele Versuche, andere aus der Kanalkodierung bekannte Kodes in der Steganographie zu etablieren um die Einbettungseffizienz zu verbessern.[2,3,4]

Trelliskodes sind in der Kanalkodierung gut bekannt und überzeugen vor allem durch ihre guten Fehlerkorrektureigenschaften. Miller et al. stellte eine Möglichkeit vor, diese auch für Wasserzeichen zu benutzen.[5,6] Dabei werden Trelliskodes benutzt um sehr robuste Wasserzeichen zu gestalten.

In dieser Arbeit soll untersucht werden, in wie weit sich Trelliskodes auch für die Steganographie ausnutzen lassen. In Kapitel 2 werden notwendige Grundlagen erklärt, welche für das Verständnis der weiteren Arbeit wichtig sind. Kapitel 3 stellt Verfahren vor, die im Rahmen dieser Arbeit untersucht wurden und stellt Vergleiche zu bereits bekannten Verfahren auf. In Kapitel 4 soll dann ein Resümee gezogen, sowie ein Ausblick auf weitere Forschungsmöglichkeiten gegeben werden.

[2]Dagmar Schönfeld und Antje Winkler, Embedding with syndrome coding based on BCH codes. in: MM&Sec '06: Proceeding of the 8th workshop on Multimedia and security. New York, NY, USA: ACM Press, 2006, ISBN 1–59593–493–6.

[3]J. Fridrich, M. Goljan und D. Soukal, Wet paper codes with improved embedding efficiency. Information Forensics and Security, IEEE Transactions on, 1 2006:1 ⟨URL: `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1597139`⟩.

[4]Jessica Fridrich und Tomáš Filler, Practical Methods for Minimizing Embedding Impact in Steganography. in: Security, Steganography, and Watermarking of Multimedia Contents IX part of Electronic Imaging, 28 January-1 February 2007, San Jose, USA | SPIE proceedings Volume 6505. Februar 2007.

[5]M.L. Miller, G.J. Doërr und I.J. Cox, Applying informed coding and embedding to design a robust high-capacity watermark. Image Processing, IEEE Transactions on, 13 Juni 2004:6.

[6]Lin Lin et al., An efficient algorithm for informed embedding of dirty-paper trellis codes for watermarking. in: ICIP (1). 2005 ⟨URL: `http://dblp.uni-trier.de/db/conf/icip/icip2005-1.html\#LinDCM05`⟩.

2 Grundlagen

2.1 Allgemeine Grundlagen

Ein Wort ist ein Vektor von Bits. Das Nullwort wird mit $\mathbf{0}_2$ bezeichnet und steht für einen Vektor $(0 \dots 0)$ beliebiger Länge. Seien a, b Wörter. Das Hamminggewicht $w_H(a)$ ist die Anzahl der von 0 verschiedenen Bits von a. Die Länge von a wird mit $|a|$ dargestellt und bezeichnet die Anzahl der Nullen und Einsen des Wortes. Mit $a \oplus b$ ist die Addition zweier Wörter definiert durch die modulare Addition der i-ten Stellen modulo 2 ($a_i + b_i \bmod 2; i = 0, 1, \dots, |a| - 1$). Die Hammingdistanz bzw. der Hammingabstand von a und b ist die Anzahl der unterschiedlichen Bits von a und b ($d_H(a, b) = w_H(a \oplus b)$).

2.2 Grundlagen der Kanalkodierung

Die Kanalkodierung verfolgt den Zweck, Bitketten gezielt Redundanz hinzuzufügen, damit sie über einen fehlerbehafteten Kanal übertragen und anschließend gestörte Bits erkannt bzw. korrigiert werden können. Der grundsätzliche Ablauf einer Übertragung ist in Abbildung 2.1 dargestellt. Dabei wird ein *Quellkodewort* (kurz: *Quellwort*)

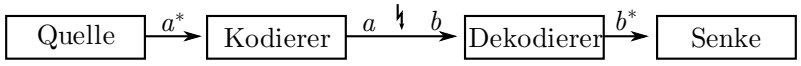


Abbildung 2.1: Ablauf einer digitalen Übertragung

von einem Kodierer in ein *Kanalkodewort* (kurz: *Kanalwort*) transformiert. Der Kodierungsprozess für einen bestimmten Kode K soll mit $\text{code}_K(a^*) = a$ bezeichnet werden. Allgemein schreiben wir für einen Kodierungsprozess mit beliebigem Kode $\text{code}(a^*) = a$. Das Kanalkodealphabet eines Kodes K bezeichnet alle Worte, die durch $\text{code}_K(a^*)$ erzeugt werden können. Während der Übertragung über einen Kanal kann das Kanalkodewort a zu einem anderen Wort b verfälscht werden. Der Dekodierer versuchte eventuell vorhandene Fehler zu korrigieren und transformiert das Wort zurück in ein Quellkodewort. Äquivalent zu den Kodierungsfunktionen $\text{code}_K(a^*)$ und $\text{code}(a^*)$ sind die Dekodierungsfunktionen $\text{decode}_K(b) = b^*$ bzw. $\text{decode}(b) = b^*$ definiert. Ist $a^* = b^*$, so wurde der Fehler korrigiert. Kann b nicht eindeutig dekodiert werden spricht man von Dekodierversagen. Der dritte Fall wird Falschkorrektur genannt. In diesem Fall meint der Dekodierer die Folge korrekt dekodieren zu können, allerdings ist $a^* \neq b^*$.

In der Kanalkodierung wird zwischen Blockkodes und blockfreien Kodes unterschieden. Blockkodes operieren immer auf festen Blöcken, wohingegen blockfreie Kodes Bitströme als Eingabe verarbeiten können. Kanalkodes werden durch mehrere Parameter unterschieden. Wichtig sind dabei die Koderate R und die Leistungsfähigkeit. Die Koderate macht eine Aussage darüber, wie viel Redundanz ein Kodierer einem Quellkodewort hinzufügt. Die Koderate wird bemessen indem wir Quellkodewortlänge ($l = |a^*|$) durch Kanalkodewortlänge ($n = |a|$) dividieren ($R = \frac{l}{n}$).

Die Leistungsfähigkeit von Kodierern wird oft mit der minimalen Hammingdistanz (bzw. Minimalabstand) d_{min} angegeben. Der Minimalabstand ist die kleinste Hammingdistanz, die ein Kanalkodewort von einem anderen Kanalkodewort hat (formal geschrieben: $d_{min} = \min\{d_H(a_1, a_2) | \forall a_1 \neq a_2 \in \text{Kanalkodealphabet}\}$). Kennt man den Minimalabstand eines Kodes, so kann man eine Aussage darüber treffen, wie viele Fehler der Kode erkennen und wie viele er korrigieren kann. Das ist in Abbildung 2.2 illustriert. Ein Kode kann immer $\left\lceil \frac{d_{min}-1}{2} \right\rceil$ Fehler korrigieren. Diese Größe wird auch die Fehlerkorrektur f_k eines Kodes genannt. Stellt man sich das Kanalkodealphabet graphisch vor, so kann man um die Kanalkodewörter Kugeln zeichnen, in denen alle Wörter zum Kanalkodewort korrigiert werden. Diese Kugeln werden als *Korrekturkugeln* bezeichnet.

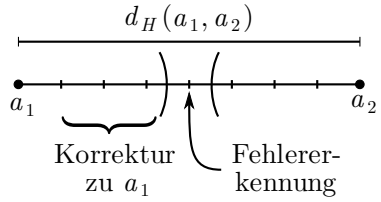


Abbildung 2.2: Graphische Darstellung des Abstandes zweier Kanalkodewörter

In der Kanalkodierung wird zwischen hard decision-Dekodierung und soft decision-Dekodierung unterschieden. Da das Signal in der Regel über einen analogen Kanal übertragen wird, wird es vor der Dekodierung auf verschiedene Quantisierungsstufen abgebildet. Versteht der Dekodierer nur 2 Quantisierungsstufen (0 oder 1) so spricht man von einer hard decision-Dekodierung, andernfalls von einer soft decision-Dekodierung.

2.2.1 Faltungskodes

Faltungskodes wurden erstmals 1955 von Peter Elias vorgestellt.[1] Im Gegensatz zu den klassischen Blockkodes wie Hamming-, RM-, BCH-, oder RS-Kodes, sind die Faltungskodes blockfreie Kodes. Sie haben den Vorteil, dass Streufehler besser korrigiert werden können, versagen dafür eher bei Bündelfehlern. Durch Kodeverkettung kann

[1] Peter Elias, Coding for Noisy Channels. IRE Conv. Rec. 4 1955.

dieser Nachteil ausgeglichen werden. Hier soll ein kurzer Überblick über Faltungskodes gegeben werden. Für einen tiefergreifenden Einblick sei exemplarisch auf Klimant et al.[2] verwiesen.

Faltungskodes lassen sich auf mehrere Arten beschreiben. Am gängigsten ist die Beschreibung durch ein Schieberegister. Ein allgemeines Schaltbild für so ein Schieberegister ist in Abbildung 2.3 gegeben.

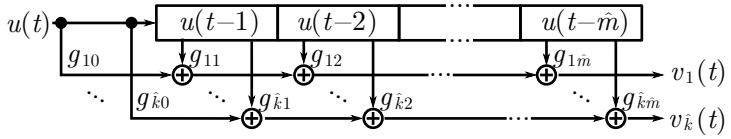


Abbildung 2.3: Allgemeines Schieberegisterschaltbild eines Faltungskodierers mit der Generatormatrix $G = \begin{pmatrix} g_{10} \cdots g_{1\hat{m}} \\ \vdots \ \ddots \ \vdots \\ g_{\hat{k}0} \cdots g_{\hat{k}\hat{m}} \end{pmatrix}$

Jedes Schieberegister besitzt einen Eingang $u(t)$ und $\hat{k}$ Ausgänge $v_1(t) \dots v_{\hat{k}}(t)$. Des Weiteren wird es durch die Anzahl der Speicherelemente bestimmt. Diese Größe wird auch als das Gedächtnis $\hat{m}$ bezeichnet.[3] Zwei Faltungskodierer gleichen Gedächtnisses unterscheiden sich nur in der Art wie die Speicherelemente mit dem Ausgang verknüpft sind. Für $1 \leq i \leq \hat{k}$; $0 \leq j \leq \hat{m}$ mit $g_{ij} \in \{0, 1\}$ wird festgelegt, ob ein Speicherelement (bzw. bei $j = 0$ der Eingang) auf den i-ten Ausgang einwirkt. Ein Faltungskodierer mit dem Gedächtnis $\hat{m}$ und $\hat{k}$ Ausgängen definiert eine Generatormatrix der Dimension $\hat{k} \times (\hat{m} + 1)$:

$$G = \begin{pmatrix} g_{10} \cdots g_{1\hat{m}} \\ \vdots \ \ddots \ \vdots \\ g_{\hat{k}0} \cdots g_{\hat{k}\hat{m}} \end{pmatrix}.$$

Diese Generatormatrix kann auch in Kurzschreibweise als $\hat{k}$-Tupel geschrieben werden. Hierbei wird jede Zeile als Oktalzahl interpretiert.[4]

Ein Ausgangsbit $v_i(t)$ zum Zeitpunkt t berechnet sich durch:

$$v_i(t) = \sum_{j=0}^{\hat{m}} u(t - j) \cdot g_{ij}.$$

[2] Herbert Klimant, Rudi Piotraschke und Dagmar Schönfeld, Informations- und Kodierungstheorie. Teubner, 2006, ISBN 3835100424, S. 199 ff.

[3] $\hat{m}$ für engl. memory

[4] Beispiel: Für die Generatormatrix $\begin{pmatrix} 101 \\ 111 \end{pmatrix}$ würde in Kurzschreibweise $(5_8, 7_8)$ geschrieben.

Mittels Matrizenmultiplikation kann der Ausgabevektor $v(t)$ berechnet werden:

$$v(t) = \begin{pmatrix} v_1(t) \\ v_2(t) \\ \vdots \\ v_{\hat{k}}(t) \end{pmatrix} = G \cdot \begin{pmatrix} u(t) \\ u(t-1) \\ \vdots \\ u(t-\hat{m}) \end{pmatrix}.$$

Die Koderate eines Faltungskodierers kann unter Kenntnis der Anzahl an Ausgängen $\hat{k}$ mit der Formel $R = \frac{1}{\hat{k}}$ berechnet werden.

Statt durch ein Schieberegister, kann ein Faltungskodierer auch als deterministischer Zustandsautomat dargestellt werden. Der Startzustand solch eines Zustandsautomaten ist äquivalent zur Anfangsinitialisierung des Schieberegisters. Eine Besonderheit des Automaten ist, dass der Folgezustand eines Zustandes nicht von der Generatormatrix abhängt.

Beispiel 1 Angenommen sei der Faltungskodierer mit der Generatormatrix $G = (5_8, 7_8)$ und dem Gedächtnis $\hat{m} = 2$. Da sich der Folgezustand $z(t+1)$ aus $u(t)u(t-1)$ ergibt, hat jeder Faltungskodierer unabhängig von seiner Generatormatrix folgende Folgezustandsabbildung:

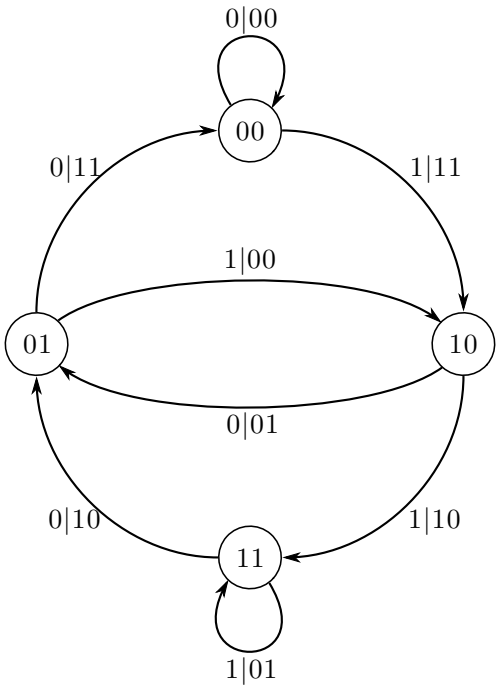
$z(t)$	$z(t+1)$	
	$u(t) = 0$	$u(t) = 1$
00	00	10
01	00	10
10	01	11
11	01	11

Faltungskodierer mit gleichem Gedächtnis unterscheiden sich also nur in der Ausgabe $v_1(t)$ und $v_2(t)$.

Die Tabelle für die Ausgaben $v(t)$ lautet für G:

$z(t)$	$v(t)$	
	$u(t) = 0$	$u(t) = 1$
00	00	11
01	11	00
10	01	10
11	10	01

Mit Hilfe beider Tabellen kann nun ein Zustandsgraph für G gezeichnet werden:



□

Betrachten wir noch einmal den Faltungskodierer G aus Beispiel 1. Wenn wir die 2 abgehenden Kanten eines Zustandes im Zustandsgraph betrachten fällt auf, dass die Ausgabesequenzen immer komplementär sind. Dafür stelle man sich einfach vor, die Beschriftung der Kanten im Zustandsgraph wäre am Ursprung eines jeden Pfeils. Dann würden an jedem Zustand zwei komplementäre Ausgabebeschriftungen stehen. In der Tabelle für die Ausgaben $v(t)$ ist dieser Effekt zu sehen, wenn wir die beiden Ausgaben (für $u(t) = 0$ und $u(t) = 1$) für einen Zustand $z(t)$ betrachten. Aufgrund der Tatsache, dass dieses Phänomen für das Paar der abgehenden Kanten auftritt, soll es *komplementäre Ausgänge* genannt werden.

Der gleiche Effekt, dass die Ausgabesequenzen komplementär sind, kann an anderer Stelle beobachtet werden. Man stelle sich dafür vor, die Beschriftung der Kanten im Zustandsgraph wäre immer an den Pfeilspitzen. Auch in diesem Fall würden an einem Zustand immer zwei komplementäre Ausgabebeschriftungen stehen. Da in diesem Fall immer zwei in einen Zustand eingehende Kanten betrachtet wurden, soll dieses Phänomen *komplementäre Eingänge* genannt werden.

Die Ursache für diese Effekte soll nachfolgend geschildert werden: Angenommen, der Eingang $u(t)$ geht direkt in die Berechnung eines Ausgangs $v_i(t)$ mit ein, daher $g_{i0} = 1$ oder $v_i(t) = u(t) \oplus \sum_{j=1}^{\hat{m}} g_{ij} \cdot u(t-j)$. Hierbei ist ersichtlich, dass das Ausgabebit $v_i(t)$ invertiert wird, wenn die Eingabe $u(t)$ invertiert wird. Hängen nun alle Ausgänge $v_0(t) \dots v_{\hat{k}}(t)$ in dieser Form vom Eingang ab, so sind die Ausgabesequenzen an zwei abgehenden Kanten komplementär, es kommt also zu komplementären Ausgängen.

Ähnlich können die komplementären Eingänge erklärt werden. Dazu ist es wichtig, dass wir analysieren, welche 2 Zustände je in einen gleichen Folgezustand führen. Betrachten wir die zwei Zustände $z_0(t)$ und $z_1(t)$ mit $z_i(t) = u(t-1) \dots u(t-\hat{m}-1)\, i$. Aufgrund der Tatsache, dass der Zustandsautomat aus einem Schieberegister hergeleitet wurde, muss der Folgezustand $z(t+1)$ dieser beiden Zustände der gleiche sein (das letzte Bit wird aus dem Schieberegister herausgeschoben). Geht nun das letzte Speicherelement $u(t-\hat{m})$ mit in die Berechnung für alle Ausgänge $v_i(t)$ ein (daher $g_{i\hat{m}} = 1$ oder $v_i(t) = \left(\sum_{j=0}^{\hat{m}-1} g_{ij} \cdot u(t-j)\right) \oplus u(t-\hat{m})$), so lässt sich auf die gleiche Weise argumentieren wie bei den komplementären Ausgängen. Da das letzte Speicherelement $u(t-\hat{m})$ zweier eingehender Kanten immer komplementär ist, muss die Ausgabe $v_i(t)$ an zwei eingehenden Kanten komplementär sein.

Bei der Dekodierung spielt das Wissen um den Start und den Endzustand eine wichtige Rolle. Um die Leistungsfähigkeit des Faltungskodierers zu erhöhen wird in der Praxis häufig versucht, im gleichen Zustand aufzuhören, in dem man die Übertragung angefangen hat. Es gibt zwei Möglichkeiten, diese Eigenschaften zu erreichen:

Die erste ist, dass nach dem Kodieren des Kodewortes noch so viele Stellen nachgeschoben werden, wie der Kodierer Speicherelemente hat. Wird beispielsweise eine Kodierung im Zustand 01 begonnen, so müssen nach erfolgter Kodierung noch eine 10 nachgeschoben werden, damit der Kodierer wieder seinen Startzustand erreicht. Natürlich verringert sich so die Koderate. Diese Art der Kodierung nennt sich *Terminierung*. Um das System einfach zu halten wird häufig im Nullzustand begonnen.

Die zweite Möglichkeit umgeht den Nachteil der Koderatenverschlechterung. Hierbei wird die Anfangsinitialisierung gleich den letzten $\hat{m}$ Stellen des Kodeworts gesetzt. Dadurch ist Anfangs- und Endzustand automatisch gleich. Dieser Kodierungsmodus wird *Tail-Biting* genannt.

Es gibt noch einen dritten Kodierungsmodus, welcher für die Kanalkodierung allerdings weniger Bedeutung hat, da hier die Fehlerkorrektur nicht so hoch ist. In diesem verzichtet man auf Kenntnis des Endzustandes und legt sich lediglich auf einen Startzustand fest. Nach Kodierung eines Wortes stoppt man in dem Zustand, in dem man sich befindet und setzt den Automaten lediglich für das nächste Kodewort zurück. Dieser Modus wird als *Truncation* bezeichnet.

Eine Darstellungsform, welche für die Dekodierung große Bedeutung hat, ist das Trellisdiagramm. Hierbei werden die Zustände dargestellt, die der Automat im Verlauf einer Übertragung erreichen kann. Ähnlich eines Baumdiagramms wird die Ausgabe des Automaten samt seiner Zustände abgetragen. Im Unterschied zum Baumdiagramm werden Knoten zusammengelegt, in denen der Automat zu einem Zeitpunkt t im gleichen Zustand ist. Ein Beispiel soll das Trellisdiagramm veranschaulichen:

Beispiel 2 Angenommen, wir wollen das Quellkodewort $a^* = (101)$ mit dem Kodierer aus dem vorangegangenen Beispiel ($G = (5_8, 7_8)$) kodieren. Dafür verwenden wir Terminierung in den Nullzustand. Das zu a^* zugehörige Kanalkodewort a lautet (11 01 00 01 11). Wir wollen davon ausgehen, dass kein Fehler auftritt ($a = b$). Da

der Empfänger ein Kodewort der Länge 10 empfängt, weiß er durch Kenntnis des verwendeten Kodierers, dass 5 Schritte für die Kodierung verwendet wurden. Ebenfalls ist ihm bekannt, dass im Nullzustand begonnen wurde und dass in den letzten beiden Schritten eine 0 kodiert wurde. Mit diesem Wissen kann er das Trellisdiagramm konstruieren wie es in Abbildung 2.4 zu sehen ist.

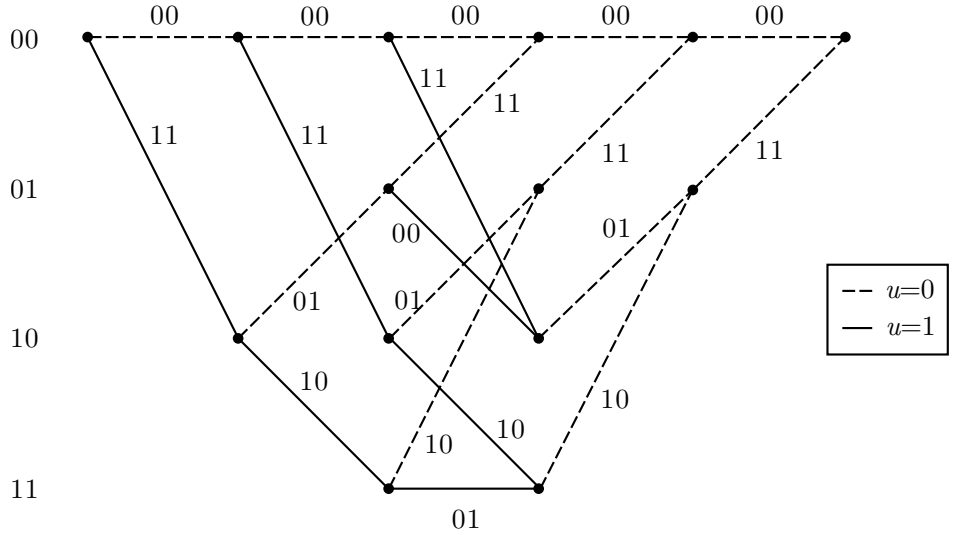
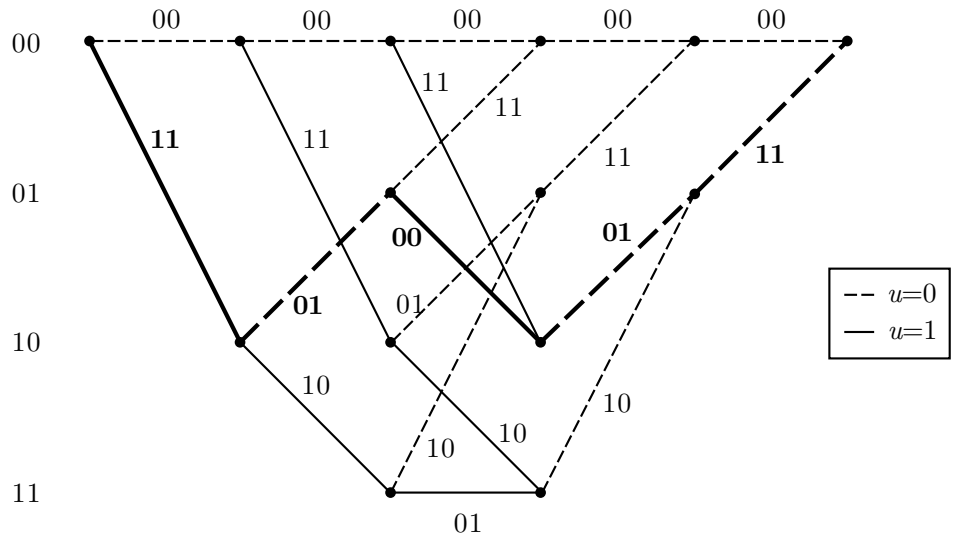


Abbildung 2.4: Trellisdiagramm mit 5 Schritten, Terminierung in den Nullzustand und der Generatormatrix $G = (5_8, 7_8)$

Die Linienart kennzeichnet hierbei die Eingabe $u(t)$. Für $u(t) = 1$ sind durchgezogene Linien dargestellt und für $u(t) = 0$ gestrichelte Linien. Die Beschriftung an den Linien kennzeichnet die Ausgabe $v(t)$. Der Empfänger kann nun anhand der Übereinstimmung der Ausgabe $v(t)$ herausfinden, welches das ursprüngliche Kodewort a^* war. In Abbildung 2.5 ist diese Dekodierung dargestellt. Der Weg des Quellkodewortes ist fett hervorgehoben.

In Abbildung 2.4 ist schon zu sehen, dass der Kodierer nach $\hat{m}$ Schritten jeden Zustand erreichen kann. Von diesem Zeitpunkt wiederholt sich das Trellisdiagramm bis zu dem Zeitpunkt an dem die Terminierungsbits nachgeschoben werden. Für eine Beschreibung eines Kodierers reicht daher auch ein verkürztes Trellisdiagramm, welches in Abbildung 2.6 zu sehen ist. □

In der Praxis ist die Dekodierung etwas schwieriger als im vorangegangenen Beispiel. Da das Kanalkodewort über einen gestörten Kanal übertragen wird, wird es verfälscht. Daher kann nicht einfach verglichen werden, wo das gesuchte Wort im Trellisdiagramm auftritt, sondern es muss nach dem Kodewort gesucht werden, was der Empfangsfolge



b	11	01	00	01	11
b^*	1	0	1	0	0

Abbildung 2.5: Dekodierung des Empfangsvektors $b = (11\,01\,00\,01\,11)$ ohne Fehlerkorrektur und der Generatormatrix $G = (5_8, 7_8)$. Der Weg des gesuchten Quellkodewortes ist fett hervorgehoben.

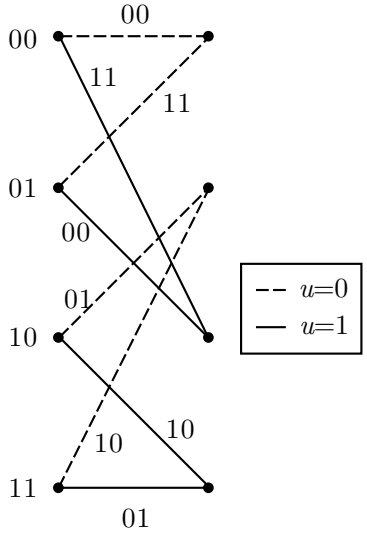


Abbildung 2.6: Verkürztes Trellisdiagramm für einen Faltungskodierer der Generatormatrix $G = (5_8, 7_8)$

am nächsten kommt. Ein solcher Algorithmus wurde 1967 von A. Viterbi vorgestellt.[5] Dieser soll hier an einem Beispiel erklärt werden.

Beispiel 3 Gehen wir von dem Kode des letzten Beispiels aus. Das Kanalkodewort $a = (11\,01\,00\,01\,11)$ soll diesmal an 3 Stellen verfälscht worden sein. Unsere Empfangsfolge lautet $b = a \oplus (01\,00\,10\,00\,10) = (10\,01\,10\,01\,01)$. Wie im letzten Beispiel stellen wir das Trellisdiagramm auf und vergleichen unsere Empfangsfolge mit den Pfaden. Dieses Mal stimmt allerdings kein Pfad mit der Empfangsfolge überein, deshalb schreiben wir überhalb der Zustände die Zahl der Abweichungen von dem Empfangswort. Das ist in Abbildung 2.7 gezeigt. Im ersten Schritt in Abbildung 2.7

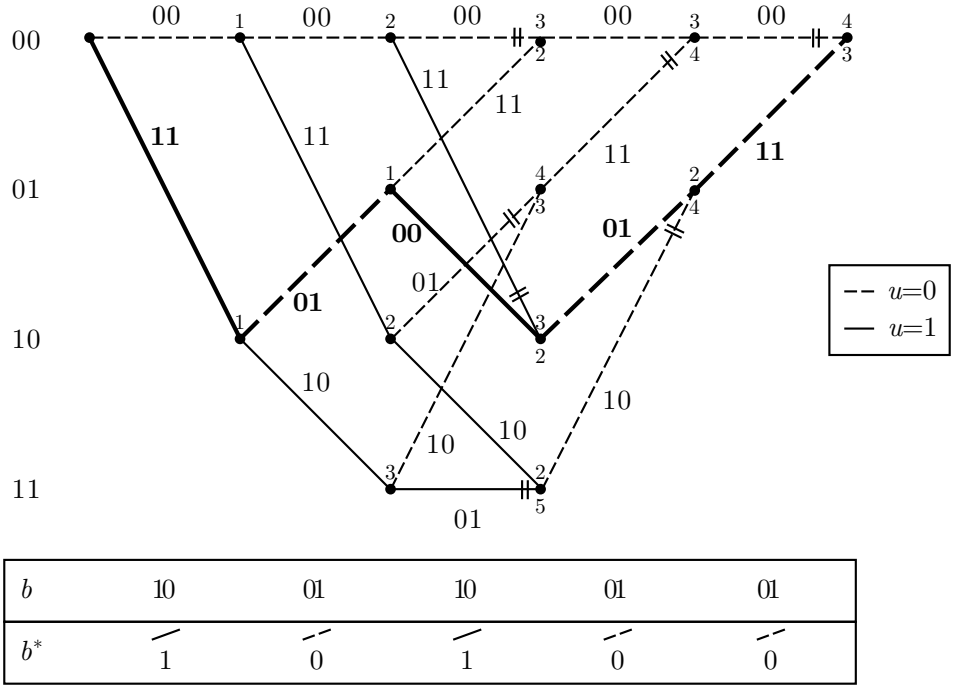


Abbildung 2.7: Dekodierung des Empfangsvektors $b = (10\,01\,10\,01\,01)$ ohne Fehlerkorrektur und der Generatormatrix $G = (5_8, 7_8)$. Der Weg des gesuchten Quellkodewortes ist fett hervorgehoben.

ist die Kantenbeschriftung vom Nullzustand 00 bzw. 11. Unsere Empfangsfolge für diesen Schritt ist 10. Da der Abstand von 00 bzw. 11 zu 10 in beiden Fällen 1 ist, schreiben wir über den Zustand im nächsten Schritt eine 1.

Im folgenden Schritt wird über den Zustand wieder die Differenz zur aktuellen Empfangsfolge geschrieben aber diesmal noch der Wert aus dem vorherigen Pfad

[5] A. J. Viterbi, Error Bounds for Convolutional Codes and an Asymptotically Optimal Decoding Algorithm. IEEE Trans. Inform. Theory, 13 April 1967:2.

addiert. Für den Zustandsübergang der oberen Zeile (von Zustand 00 zu 00) wäre das $d_H(00, 01) + 1 = 2$.

Interessant ist der nächste Schritt. Hier befinden sich an jedem Zustand 2 Zahlen, eine für jeden Pfad. Von diesen Zahlen wird nun das Minimum genommen und der Pfad mit der größeren weggestrichen.

Wurden alle Gewichte eingetragen, so kann der Pfad mit dem kleinsten Gewicht zurückverfolgt und an diesem b^* abgelesen werden. □

Für weiterführende Literatur sei hier auf Bossert[6] sowie Klimant et al.[7] verwiesen.

Die Leistungsfähigkeit eines Faltungskodierers wird häufig mit der *freien Distanz* d_f angegeben. Ähnlich zu den Blockkodes macht diese Angabe eine Aussage über die Anzahl der Fehler, die der Kode korrigieren kann, allerdings ist dieser Wert bei Faltungskodes schwerer zu bestimmen. Eine Methode zum Finden guter Faltungskodes samt einiger Tabellen stellte Palazzo vor.[8]

2.2.2 Rekursive Faltungskodes

Rekursive Faltungskodierer stechen durch ihre unendliche Einflusslänge hervor und haben für Turbokodes große Bedeutung. In der Kodierungstheorie werden rekursive Faltungskodierer ausschließlich als rekursiv systematische Faltungskodierer verwendet. In Abbildung 2.8 ist ein rekursiv systematischer Faltungskodierer mit der Genera-

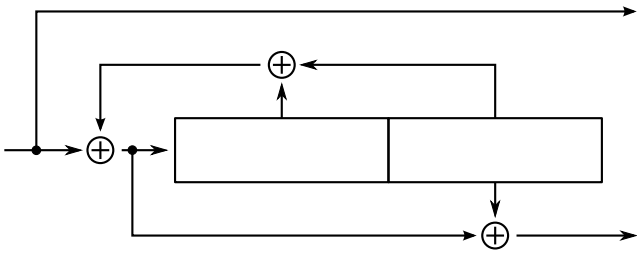


Abbildung 2.8: Rekursiv systematischer Faltungskodierer mit der Generatormatrix $G = (4_8, \frac{5_8}{7_8})$

tormatrix $G = (4_8, \frac{5_8}{7_8})$ zu sehen. Dabei steht die 4_8 in der Generatormatrix für den systematischen Ausgang und die $\frac{5_8}{7_8}$ dafür, dass der Ausgang mit der Belegung 5_8, mit einer 7_8-Rekursion versehen ist.

[6] Martin Bossert, Kanalcodierung (Teubner Informationstechnik). Teubner Verlag, 1998, ISBN 3519161435, S. 286 ff.

[7] Klimant, Piotraschke und Schönfeld, *op. cit.*, S. 212 ff.

[8] Jr. Palazzo, R.P., A network flow approach to convolutional codes. Communications, IEEE Transactions on, 43 Februar -März -April 1995:234.

2.2.3 Trelliskodes

Unter Trelliskodes versteht man alle in einem Trellisdiagramm dargestellten Kodes. Das sind natürlich zum einen die Faltungskodes wie in Abschnitt 2.2.1 gezeigt. Es lassen sich aber auch andere Kodes in einem Trellisdiagramm darstellen. Wie sich Blockkodes mit einem Trellisdiagramm ausdrücken lassen wurde schon 1974 von Bahl et al. gezeigt.[9] Der Vorteil des Trellisdiagramms liegt darin, dass dann soft decision-Dekodierung angewendet und damit eine höhere Leistungsfähigkeit erreicht werden kann.

Eine weitere Möglichkeit einen Trelliskode zu erzeugen ist, ein Trellisdiagramm nach belieben zu spezifizieren. Dabei kann die sowohl die Kantenbeschriftung als auch die Zuordnung von Zustand zu Folgezustand frei gewählt werden. Es ist sogar denkbar, dass der Folgezustand sowie die Kantenbeschriftung in jedem Taktzeitpunkt verschieden sind (was eine Darstellung als verkürztes Trellisdiagramm unmöglich machen würde). Für die Kanalkodierung spielen solche Zufallstrelliskodes allerdings keine große Rolle, da sich hier nur wenig über den Minimalabstand und die Fehlerkorrektur sagen lässt.

2.3 Steganographische Grundlagen

Steganographie ist die Kunst, Nachrichten zu verstecken. Das Wort „Steganographie“ ist aus dem Altgriechischen von „schützend, verdeckt“ und „schreiben“ abgeleitet. Anders als in der Kryptographie kommt es hierbei nicht darauf an einem Angreifer Nachrichteninhalte zu verbergen, sondern das Vorhandensein von Nachrichten zu verstecken.[10] Im Gegensatz zur Kryptographie, die erst als gebrochen zählt, sobald man den Inhalt einer Nachricht entschlüsseln kann, zählt ein steganographisches Verfahren als gebrochen, sobald die Existenz einer Nachricht erkannt wird.

Schon in der Vergangenheit wurde versucht, mittels eines Trägermediums eine Nachricht zu übermitteln. So wurden Nachrichten beispielsweise an Tiere verfüttert. Der Empfänger konnte das Tier dann schlachten und diese Nachricht wieder gewinnen. Teilweise wurde mit „unsichtbarer Tinte“ eine Nachricht auf ein Blatt Papier geschrieben. Mittels chemischer Substanzen konnte diese Nachricht anschließend lesbar gemacht werden.

Auch in der modernen Steganographie bedient man sich eines Trägermediums. Heute werden meist digitalisierte Bilder, Audio- oder Videodateien verwendet. Ein

[9] L. Bahl et al., Optimal decoding of linear codes for minimizing symbol error rate. IEEE Transactions on Information Theory, 20 Mar. 1974:2.

[10] Allerdings ist natürlich normalerweise das Vorhandensein von Nachrichten erkannt, wenn deren Nachrichteninhalt erkannt ist. Des Weiteren ist es für die Steganographie meist günstig, Redundanzen aus Nachrichten zu entfernen, was durch Kryptographie erreicht werden kann, aber auch durch eine geeignete Kodierung (z. B. Shannon-Fano oder Huffman Kodierung).

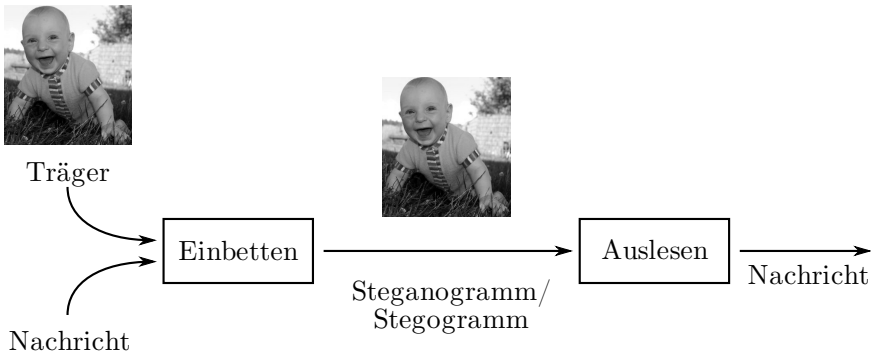


Abbildung 2.9: Ablauf einer steganographisch gesicherten Nachrichtenübertragung

prinzipieller Ablauf einer steganographisch gesicherten Nachrichtenübertragung ist in Abbildung 2.9 gezeigt.

Um einen Einblick zu bekommen, wollen wir nachfolgend die einfachste Methode, die LSB-Steganographie[11] mit Bildern, erklären: Gehen wir dazu von einem Graustufenbild als Trägermedium aus, welches für jeden Bildpunkt einen Grauwert gespeichert hat. Das niederwertigste Bit eines Grauwertes hat dabei auf dessen Helligkeit am wenigsten Einfluss. Überschreibt man dieses Bit mit seinen eigenen Informationen, ist die dadurch resultierende Veränderung für das menschliche Auge kaum bzw. gar nicht sichtbar, allerdings kann ein Empfänger die niederwertigsten Bits auslesen und wieder zu einer Nachricht zusammensetzen.

Beispiel 4 Angenommen wir wollen die Nachricht $m = (1101)$ in ein Graustufenbild einbetten. Die ersten 4 Grauwerte $(x_0, \ldots, x_3)$ unseres Trägerbildes seien 32 33 32 32. Die niederwertigsten Bits dieser 4 Punkte $(b(x_0), \ldots, b(x_3))$ lauten daher 0 1 0 0. Diese werden auch als Träger bezeichnet $(c = (0100))$[12]. Von diesen 4 Bits stimmen das erste und das letzte nicht mit unserer Nachricht überein, diese Bildpunkte müssen also geändert werden. Statt der ursprünglichen 4 Punkte übertragen wir also **33** 33 32 **33**. Das wird auch als das Steganogramm bezeichnet und mit x' notiert (bzw. c' für $b(x'))$. □

Diese einfache Methode ist allerdings durch statistische Tests sehr leicht nachweisbar. Deshalb wird versucht, die Anzahl der Änderungen pro Nachrichtenbit zu verringern. Das kann z. B. dadurch passieren, dass die niederwertigsten Bits mehrerer Bildpunkte zusammenaddiert werden und eines von ihnen geändert wird, falls die Summe ungleich dem einzubettenden Bit ist. Diese Einbettungsmethode wird auch als *Einbetten mit Paritätskodierung* bezeichnet.

[11] LSB, engl. least significant bit, das niederwertigste Bit
[12] c von engl. cover

Beispiel 5 Wir wollen die gleiche Nachricht $m = (1101)$ einbetten. Die folgende Tabelle zeigt die ersten 16 Grauwerte des Trägerbildes und die sich daraus ergebenden niederwertigsten Bits. Die letzte Spalte ist eine Parität der 4 niederwertigsten Bits.

Grauwerte im Träger (x_i)	LSB ($b(x_i)$)	Parität
32 33 32 32	0 1 0 0	1
30 29 29 28	0 1 1 0	0
30 31 93 93	0 1 1 1	1
31 31 93 94	1 1 1 0	1

Vergleicht man den Paritätsvektor (1011) mit der Nachricht, so sind die mittleren Bits unterschiedlich. Daher muss ein Grauwert in der 2. und ein Grauwert in der 3. Zeile geändert werden. Die übertragenen Grauwerte könnten daher so aussehen:

Grauwerte im Steganogramm (x'_i)	LSB ($b(x'_i)$)	Parität
32 33 32 32	0 1 0 0	1
31 29 29 28	**1** 1 1 0	**1**
31 29 29 28	**1** 1 1 1	**0**
31 31 93 94	1 1 1 0	1

Natürlich hätten wir statt der Grauwerte in der ersten Spalte auch andere Grauwerte ändern können, um auf die gleiche Parität zu kommen.

Ähnlich zu den Funktionen für die Kodierung und Dekodierung in der Kanalkodierung können diese Funktionen hier wieder definiert werden. K steht für den steganographischen Kodierer bzw. Dekodierer. Dabei benötigt die Funktion $\text{code}_K()$ nun allerdings 2 Parameter, die Nachricht selbst, sowie den Teil des Trägermediums der Länge n, in den die Nachricht eingebettet werden soll. Die Ausgabe ist das Steganogramm ($\text{code}_K(m, x) = x'$). Die Funktion $\text{decode}_K()$ bekommt nun statt der Empfangsfolge b als Eingabe das Steganogramm und gibt die Nachricht aus ($\text{decode}_K(x') = m$). Die vorangegangen Methoden hätte man so mit $\text{code}_{LSB}(m, x)$, $\text{code}_{Par}(m, x)$ und $\text{decode}_{LSB}(x')$, $\text{decode}_{Par}(x')$ formal definieren können. Da viele Einbettungsalgorithmen nicht die Grauwerte benötigen, sondern mit den niederwertigsten Bits der Grauwerte auskommen, wollen wir vereinfacht $\text{code}_K(m, c) = c'$ bzw. $\text{decode}_K(c') = m$ schreiben.

An den vorangegangenen zwei steganographischen Algorithmen kann man zwei Bewertungskriterien erkennen. Die Einbettungsrate und die durchschnittliche Kippzahl. Für die nachfolgende Notation wollen wir davon ausgehen, dass in jedem Einbettungsschritt k Bits der Nachricht m in n Bits Trägermedium eingebettet werden können.

Die *Einbettungsrate* α ist ein ähnliches Maß wie die Koderate in der Kodierungstheorie. Sie ergibt sich, indem die Anzahl der eingebetteten Bits durch die Anzahl

der benötigten Bits im Trägermedium geteilt wird ($\alpha = \frac{k}{n}$). In Beispiel 5 wurden 4 Bit verwendet um ein Bit zu verstecken. Die Einbettungsrate ist für dieses Beispiel daher $\frac{1}{4}$. Wird die Parität über n Bits gebildet, so ist die allgemeine Einbettungsrate für die Paritätskodierung $\frac{1}{n}$.

Die *durchschnittliche Kippzahl* R_a erhält man indem berechnet wird, wie viele Bits pro Wort im Mittel geändert werden müssen, unter der Annahme, alle möglichen Nachrichten träten gleich häufig auf. Für die simple LSB-Steganographie ist dies $\frac{1}{2}$. Es gibt hier nur 2 Möglichkeiten. Entweder die Zahl, in der ich etwas verstecken will, ist gerade oder ungerade. Je nachdem, welches Bit ich einbetten möchte, muss ich im Mittel in 50% der Fälle das niederwertigste Bit kippen. Wird die Paritätskodierung verwendet, so habe ich auch eine 50% Chance, dass die Parität mit dem Bit übereinstimmt, welches ich einbetten möchte. Die durchschnittliche Kippzahl ist hier also auch $\frac{1}{2}$.

Die *Einbettungseffizienz* e berechnet sich aus der Länge des pro Schritt einzubettenden Wortes dividiert durch die durchschnittliche Kippzahl

$$e = \frac{k}{R_a} \tag{2.1}$$

und ist damit die Anzahl der Einbettungsbits je Änderung. Da in den vorangegangenen Beispielen die durchschnittliche Kippzahl in beiden Fällen $\frac{1}{2}$ war und pro Schritt je 1 Bit eingebettet werden konnte, ist die Effizienz beider Verfahren 2.

2.4 Kanalkodierung und Steganographie

Eine einfache Methode Kanalkodes für die Steganographie auszunutzen wurde 1998 von Ron Crandall veröffentlicht.[13] Sie nutzt u. a. die aus der Kodierungstheorie gut bekannten Hammingkodes[14] und ist unter dem Namen *Matrixkodierung* bekannt geworden. Es kann ein beliebiges steganographisches System zur Einbettung genommen werden.[15]

An dieser Stelle sollen die Begriffe Kodierung und Einbettung besser voneinander abgegrenzt werden. Einbettung bezeichnet den Vorgang, in dem die Daten im Träger verändert werden, Kodierung bezeichnet die Vorverarbeitung der Daten. Deutlich wird das hier, wo mit der Matrixkodierung mehrere code() Funktionen definiert werden können. Beispielsweise könnte man $\text{code}_{Hamming+LSB}()$ oder $\text{code}_{Hamming+Par}()$ definieren. Der erste Parameter der Fußnote ($Hamming$) verweist hierbei auf die Kodierungsmethode und der zweite (LSB bzw. Par) auf die Einbettungsmethode.

[13] Ron Crandall, Some Notes on steganography. Posted on Steganography Mailing List, Dezember 1998 ⟨URL: `http://os.inf.tu-dresden.de/~westfeld/crandall.pdf`⟩.

[14] R. W. Hamming, Error detecting and error correcting codes. The Bell System Technical Journal, 29 April 1950:2.

[15] z. B. LSB-Steganographie oder Einbetten mit Paritätskodierung

Leider lassen sich die beiden Bereiche nicht immer eindeutig voneinander abgrenzen und wir werden im Folgenden noch sehen, dass es für die Einbettungseffizienz sogar günstig ist, wenn Kodierungs- und Einbettungsalgorithmus voneinander abhängig sind.

Der Einfachheit halber wollen wir im folgenden Verlauf immer LSB als Einbettungssystem verwenden, sofern nichts anderes gesagt wird.

Mittels geeigneter Kodierung wird nun versucht die Einbettungseffizienz der LSB-Steganographie zu erhöhen. Hierbei wird das Syndrom, welches vom Empfänger normalerweise dafür verwendet wird um die Nachricht zu korrigieren, zur eigentlichen Nachricht deklariert. Der Empfänger kann also wie gewohnt die Empfangsfolge mit der Kontrollmatrix multiplizieren und bekommt damit direkt die Nachricht heraus. Formal ausgedrückt bedeutet das: $\text{decode}_{Hamming}(c') = H \cdot c'^T$.

Der Sender muss daher das Trägermedium so manipulieren, dass durch Multiplikation mit der Kontrollmatrix die gewünschte Nachricht herauskommt. Die Funktion $\text{code}_{Hamming}(m, c)$ soll nachfolgend verbal beschrieben werden:

1. Die Folge im Trägermedium c wird mit der Kontrollmatrix multipliziert. Man erhält das originale Syndrom s_0.

2. Die einzubettende Nachricht m wird mit dem originalen Syndrom bitweise modulo 2 addiert ($s_1 = m \oplus s_0$).

3. Die Stelle an der in der Kontrollmatrix die Spaltenbelegung s_1 steht[16] wird im Träger gekippt.

Ein Beispiel soll diese Methode nochmals veranschaulichen:

Beispiel 6 Angenommen wir wollen $m = (101)$ mit einem $(n = 7, l = 4)$-Hammingkode in die Trägerfolge $c = (0110111)$ einbetten. Die Kontrollmatrix für diesen Kode lautet $H = \begin{pmatrix} 1111000 \\ 1100110 \\ 1010101 \end{pmatrix}$.

1. Kontrollmatrix mit Trägerfolge multiplizieren:

$$s_0 = H \cdot c^T = \begin{pmatrix} 1111000 \\ 1100110 \\ 1010101 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}$$

[16] Da die Kontrollmatrix bei Hammingkodes normalerweise durchnummeriert ist kann die Stelle direkt von s_1 abgelesen werden.

2. $s_1 = m \oplus s_0 = (101) \oplus (011) = (110)$

3. $s_1 = (110) \rightarrow 110_2 = 6_{10} \rightarrow$ die 6. Stelle (von hinten gezählt) wird gekippt: $c = (0110111) \rightarrow c \oplus (0100000) = (0010111)$

Der Empfänger liest die Folge aus und multipliziert sie mit der Kontrollmatrix:

$$m = \begin{pmatrix} 1111000 \\ 1100110 \\ 1010101 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}.$$

□

Wie aus dem Algorithmus schon hervorgeht, muss bei einer $k \times n$ Kontrollmatrix immer maximal ein Bit gekippt werden um k Bits in n Bits einzubetten. Grund dafür ist, dass der Hammingkode genau einen Fehler korrigieren kann und perfekt ist. Dadurch kann mit einer Änderung von höchstens einem Bit jede beliebige andere Nachricht eingebettet werden. Will man k Bits mittels Matrixkodierung einbetten, so gibt es 2^k mögliche Nachrichten. Das ursprüngliche Syndrom entspricht einer dieser 2^k möglichen Nachrichten ($s_0 = m$). In diesem Fall muss kein Bit geändert werden. In allen $2^k - 1$ restlichen Fällen muss ein Bit geändert werden. Dadurch lässt sich die mittlere Kipprate berechnen:

$$R_a = 2^{-k} \cdot \sum_{i=0}^{1} \binom{n}{i} \cdot i = \frac{1 \cdot 0 + (2^k - 1) \cdot 1}{2^k} = \frac{2^k - 1}{2^k}. \tag{2.2}$$

Die Effizienz eines $(n, n-k)$-Hammingkodes ist daher:

$$e = \frac{k}{R_a} = \frac{k \cdot 2^k}{2^k - 1}.$$

Auch die Einbettungsrate α ist mit Kenntnis von k leicht berechenbar:

$$\alpha = \frac{k}{n} = \frac{k}{2^k - 1}.$$

Die Effizienz verschiedener Hammingkodes ist in Abbildung 2.10 dargestellt

Nachdem Ron Crandall fehlerkorrigierende Kodes in die Steganographie einführte, wurde von verschiedener Seite versucht, die Einbettungseffizienz mit anderen aus der Kanalkodierung bekannten Kodes zu verbessern. So gab es Untersuchungen von D.

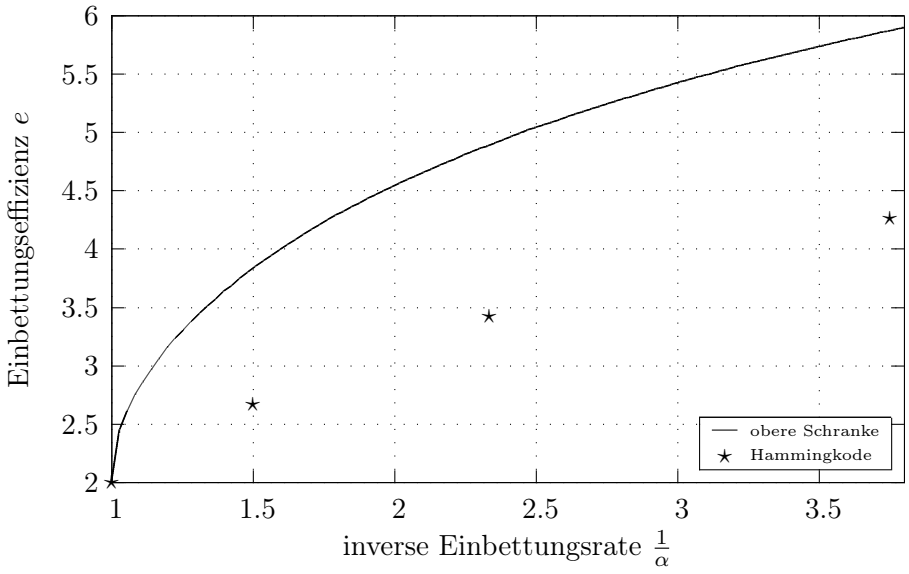


Abbildung 2.10: Effizienz verschiedener Hammingkodes

Schönfeld und A. Winkler[17], wie sich BCH-Kodes zum Kodieren für die Steganographie nutzen lassen. Fridrich et al. untersuchten Zufallskodes[18] sowie LDGM[19] Kodes.

Ein Überblick über verschiedene Kodes und ihre Effizienz ist in Abbildung 2.11 dargestellt.

Abschließend sei noch eine allgemeine Formel aufgezeigt, mit der sich die mittlere Kippzahl eines beliebigen Kodes mittels der Funktion code(m, c) berechnen lässt:

$$R_a = \frac{1}{2^{n+k}} \cdot \sum_{c=\mathbf{0}_2}^{(2^n-1)_2} \sum_{m=\mathbf{0}_2}^{(2^k-1)_2} d_H(\text{code}(m,c),c). \tag{2.3}$$

[17] Dagmar Schönfeld und Antje Winkler, Embedding with syndrome coding based on BCH codes. in: MM&Sec '06: Proceeding of the 8th workshop on Multimedia and security. New York, NY, USA: ACM Press, 2006, ISBN 1–59593–493–6.

[18] J. Fridrich, M. Goljan und D. Soukal, Wet paper codes with improved embedding efficiency. Information Forensics and Security, IEEE Transactions on, 1 2006:1 ⟨URL: `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1597139`⟩.

[19] Jessica Fridrich und Tomáš Filler, Practical Methods for Minimizing Embedding Impact in Steganography. in: Security, Steganography, and Watermarking of Multimedia Contents IX part of Electronic Imaging, 28 January-1 February 2007, San Jose, USA | SPIE proceedings Volume 6505. Februar 2007.

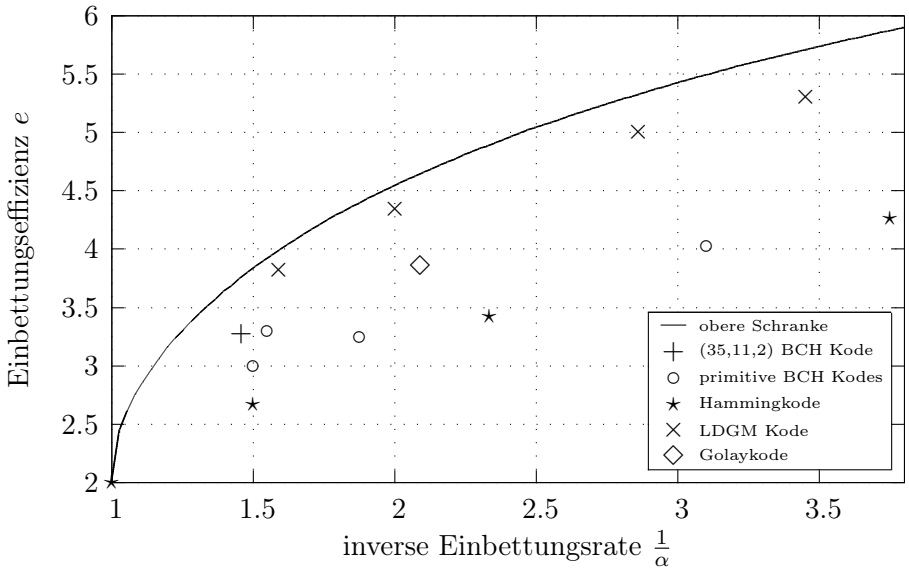


Abbildung 2.11: Effizienz verschiedener Kodes

Da diese Berechnung sehr komplex ist und für größere Kodes zu aufwendig wäre, kann man durch weglassen einer Summe einen guten Näherungswert erreichen. Die Einbettungseffizienz kann also näherungsweise für ein beliebig gewähltes m durch

$$R_a \approx \frac{1}{2^n} \sum_{c=\mathbf{0}_2}^{(2^n-1)_2} d_H(\text{code}(m,c),c) \tag{2.4}$$

und für ein beliebig gewähltes c durch

$$R_a \approx \frac{1}{2^k} \sum_{m=\mathbf{0}_2}^{(2^k-1)_2} d_H(\text{code}(m,c),c) \tag{2.5}$$

berechnet werden. Wählt man für letztere Formel $c = \mathbf{0}_2$ so erhält man zusätzlich noch die vereinfachte Formel

$$R_a \approx \frac{1}{2^k} \sum_{m=\mathbf{0}_2}^{(2^k-1)_2} w_H(\text{code}(m,\mathbf{0}_2)).$$

Beispiel 7 Es soll die durchschnittliche Kippzahl des $(7,4)$-Hammingkodes mit Matrixkodierung und Gleichung 2.5 berechnet werden. Wir wählen den Träger fest auf das Nullwort. Damit muss jede Nachricht von $m_0 = (000)$ bis $m_7 = (111)$ einmal

kodiert werden ($\text{code}(m_i, \mathbf{0}_2)$) und über das Ergebnis das mittlere Gewicht berechnet werden:

$$R_a \approx \frac{1}{8} \sum_{m=\mathbf{0}_2}^{(111)} w_H(\text{code}(m, \mathbf{0}_2)).$$

Für m_0 wird unser Kodierungsalgorithmus das Nullwort ausgeben ($\text{code}(\mathbf{0}_2, \mathbf{0}_2) = \mathbf{0}_2$). In den restlichen 7 Fällen muss ein Bit des Trägers gekippt werden, was bedeutet, dass $\text{code}(m_i, \mathbf{0}_2)$ ein Wort mit dem Gewicht 1 ausgeben wird. Die vorangegangene Gleichung kann daher mit $R_a \approx \frac{0+7 \cdot 1}{8} = 0,875$ beschrieben werden. □

An diesem Beispiel ist zu sehen, dass das approximierte Ergebnis von R_a exakt dem Ergebnis entspricht wie es mit Gleichung 2.2 berechnet werden konnte. Wir werden aber in Abschnitt 3.1 im Beispiel 8 sehen, dass das nicht immer der Fall ist.

Zhang et. al. stellten eine Verbesserung zu Blockkodes vor, in dem sie ein Einbettungsschema vorstellten, welches als Eingabe nicht nur die niederwertigsten Bits benutzt, sondern direkt mit Grauwerten rechnet.[20] Dabei werden nicht nur die Informationen aus dem niederwertigsten Bit, sondern auch aus dem zweitniederwertigsten Bit genutzt. Dieses Bit benutzen sie in der Weise, dass sie beim Einbetten nicht nur das letzte Bit überschreiben, sondern gezielt inkrementieren bzw. dekrementieren und damit auch das vorletzte Bit verändern. Ist man also nicht auf ein bestimmtes Einbettungsschema beschränkt, wie z. B. beim Einbetten mittels Inkrementieren[21] oder im F5 Algorithmus von Andreas Westfeld[22], so kann man mit diesem Schema eine Effizienzverbesserung gegenüber gewöhnlicher LSB-Steganographie erzielen. Zhang vollzieht damit die Anfangs erwähnte Vereinigung von Einbettungs- und Kodierungsschema.

Die ersten k Bit werden auf gleicher Weise dekodiert, wie es in der Matrixkodierung zu sehen war (die Funktion $b(x)$ gibt wieder von einem Grauwert x das niederwertigste Bit an):

$$\begin{pmatrix} m_1 \\ \vdots \\ m_k \end{pmatrix} = H \cdot \begin{pmatrix} b(x_1) \\ \vdots \\ b(x_n) \end{pmatrix}.$$

Zusätzlich zu diesen k Bits wird ein weiteres Bit eingebettet. Da dieses Schema $k+1$ Bits in $n+1$ Bits versteckt, wird es das $+1$ Schema genannt. Das zusätzliche Bit berechnet sich nach folgendem Schema:

[20] Weiming Zhang, Shuozhong Wang und Xinpeng Zhang, Improving Embedding Efficiency of Covering Codes. Communications, IEEE Transactions on, 11 August 2007:8.

[21] Andreas Westfeld und Andreas Pfitzmann, Attacks on Steganographic Systems. in: Andreas Pfitzmann (Hrsg.), Information Hiding. Band 1768, Springer, 1999, ISBN 3–540–67182–X.

[22] Andreas Westfeld, F5-A Steganographic Algorithm. in: IHW '01: Proceedings of the 4th International Workshop on Information Hiding. London, UK: Springer-Verlag, 2001, ISBN 3–540–42733–3.

$$m_{k+1} = \left(\left\lfloor \frac{x_1}{2} \right\rfloor + \ldots + \left\lfloor \frac{x_n}{2} \right\rfloor + x_{n+1} \right) \bmod 2.$$

Der Trick in dieser Formel ist, dass mit $\left\lfloor \frac{x}{2} \right\rfloor \bmod 2$ das vorletzte Bit ausgelesen wird. Da in den meisten Fällen ein Bit der ersten n Bits gekippt werden muss, kann hier entschieden werden, ob der Graustufenwert de- oder inkrementiert wird und auf diese Weise die 2. Formel gültig gemacht werden. Stimmt die erste Formel ohne Änderung eines der ersten n Grauwerte, so kann die Information in x_{n+1} versteckt werden, wobei dann egal ist, ob de- oder inkrementiert wird.

Da Hammingkodes im Mittel maximal ein Bit im Träger ändern ($R_a < 1$), kann hier nur das +1 Schema angewendet werden. Bei Kodes, die drei oder mehr Bits pro Einbettungsschritt kippen, kann dieses Schema auf ein +2 Schema ausgeweitet werden. Dabei werden wieder die ersten k Bits wie gewohnt dekodiert. Für die zusätzlichen 2 Bits wird folgende Gleichung verwendet:

$$(m_{k+1}, m_{k+2}) = \left(\left\lfloor \frac{x_1}{2} \right\rfloor + \ldots + \left\lfloor \frac{x_n}{2} \right\rfloor + x_{n+1} + x_{n+2} \right) \bmod 4.$$

In der Kodierung muss nun wieder darauf geachtet werden, dass statt die niederwertigsten Bits zu überschreiben, so in- bzw. dekrementiert wird, dass die zusätzlichen Bits richtig berechnet werden. Wenn zu wenige oder gar keine der ersten n Grauwerte geändert werden müssen, kann x_{n+1} bzw. x_{n+2} benutzt werden. Mit $x_i \in \{0, 1, 2\}$ und $x_{n+1} + x_{n+2} \bmod 4$ lassen sich alle 4 Werte aus Z_4 darstellen, wobei 0,1,2 einem dekrementieren, belassen bzw. inkrementieren gleichkommt.

Dieses Schema kann noch beliebig erweitert werden. Allerdings muss der unterliegende Kode dann auch von vornherein eine größere mittlere Kippzahl aufweisen. Benutzt man beispielsweise das +2 Schema für den Hammingkode, so kann man die Gleichung für (m_{k+1}, m_{k+2}) mit der einen Änderung, die der Hammingkode pro Schritt bietet, immer nur für ein Bit von den 2 zusätzlichen erfüllen. Für das andere muss immer ein zusätzliches Bit gekippt werden. Daher erreicht man für diesen Fall keine Effizienzsteigerung zur normalen LSB-Steganographie. Verglichen mit der Matrixkodierung wie sie von Crandall vorgeschlagen wurde sinkt die Effizienz dann sogar.

Werden 3 Bits zusätzlich eingebettet, so müssen 4 zusätzliche Stellen genommen werden, bei 4 Bits schon 8. Allgemein lautet die Gleichung für ein +p Schema:

$$(m_{k+1}, \ldots, m_{k+p}) = \left(\left\lfloor \frac{x_1}{2} \right\rfloor + \ldots + \left\lfloor \frac{x_n}{2} \right\rfloor + x_{n+1} + \ldots + x_{n+2^{p-1}} \right) \bmod 2^p.$$

Eine Übersicht über die Effizienzverbesserung mit diesem Schema ist in Abbildung 2.12 gezeigt.

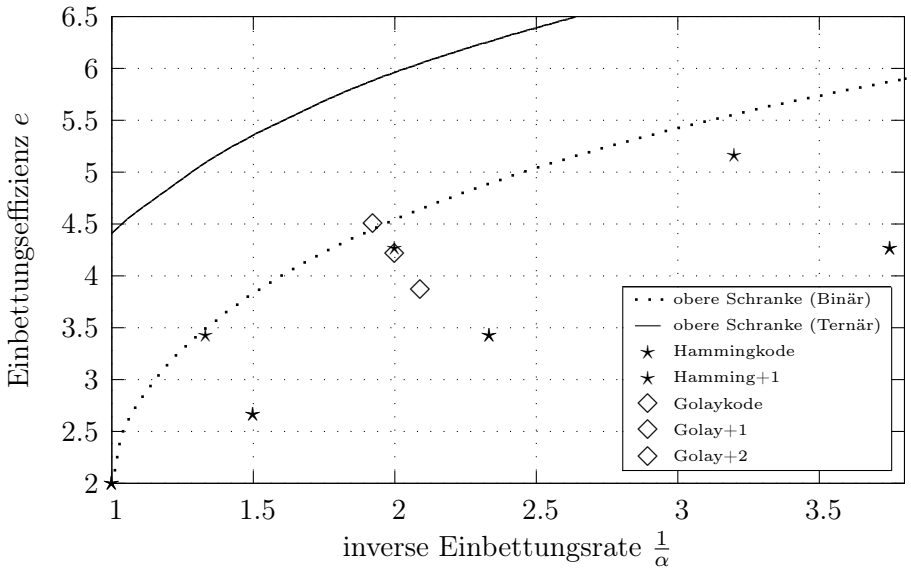


Abbildung 2.12: Verbesserung des Hamming- und Golaykodes

3 Trelliskodes

Nachdem bereits gezeigt wurde, wie sich klassische Blockkodes für die Steganographie nutzen lassen, stellt sich die Frage, ob auch Faltungskodes für die Steganographie von nutzen sein können. Die Dekodierung unterscheidet sich allerdings stark von linearen Blockkodes. Bei linearen Blockkodes findet die Kodierung einer Nachricht für die Steganographie immer über das Syndrom statt. Bei Faltungskodes hat man bei einer Dekodierung allerdings kein Syndrom, welches sich für eine Nachrichteneinbettung nutzen lässt. Der ganze Dekodierungsprozess verläuft auf eine andere Weise, so dass man sich über andere Möglichkeiten Gedanken machen muss, diesen für die Steganographie zu benutzen.

Da Faltungskodes gerade durch ihre guten Fehlerkorrektureigenschaften bestechen, ist es naheliegend, ein Kodierungsschema so aufzubauen, dass diese Eigenschaft ausgenutzt wird.

Eine einfache Methode, wie man die Fehlerkorrektur eines beliebigen Kodes für die Steganographie ausnutzen kann, ohne Rücksicht auf spezielle Eigenschaften des Kodierungs- und Dekodierungsverfahren zu nehmen, ist im Abschnitt 3.1 aufgezeigt. Eine Methode, die die Trellisstruktur nutzt und dabei auf Festlegung des Startzustandes verzichtet wird in Abschnitt 3.2 vorgestellt. Abschnitt 3.3 befasst sich dann damit, wie man die Einbettungsrate dieses Kodes noch durch Hinzunahme von Informationen aus dem Startzustand verbessern kann. In Abschnitt 3.4 wird dargestellt, wie die Einbettungseffizienz der vorgestellten Methode mit rekursiven Faltungskodierern verbessert werden kann. Ein Vergleich zum Wasserzeichenalgorithmus von Miller wird in Abschnitt 3.5 gemacht. Wie das $+1/+2$ Schema von Zhang für Trelliskodes verwendet werden kann wird in Abschnitt 3.6 erläutert. Eine abschließende Vorstellung und Diskussion weiterer Möglichkeiten, die sich leider als unpraktikabel erwiesen, findet sich in Abschnitt 3.7.

3.1 Ausnutzung der Fehlerkorrektur

Bei dieser Methode soll die Fehlerkorrektur eines Kodes genutzt werden, um weniger Bits in einem Trägermedium zu kippen, als eigentlich notwendig wären. Das Wort, welches man einbetten will, wird als das Quellkodewort eines Kodierers betrachtet ($m = a^*$). Dieses Quellkodewort wird zuerst in ein Kanalkodewort umgewandelt ($\text{code}(a^*) = a$) und anschließend mit dem Trägermedium verglichen. An allen Stellen, an denen sich das Kanalkodewort vom Trägermedium unterscheidet, können Bits im Kanalkodewort gekippt werden. Dieses gestörte Kanalkodewort wird im Träger

eingebettet. Die gestörten Stellen können in der Dekodierung wieder korrigiert werden. Daher dürfen natürlich nicht mehr Bits gestört werden, als beim Dekodierungsprozess wieder korrigiert werden können. Eine schematische Darstellung dieser Methode ist in Abbildung 3.1 gegeben.

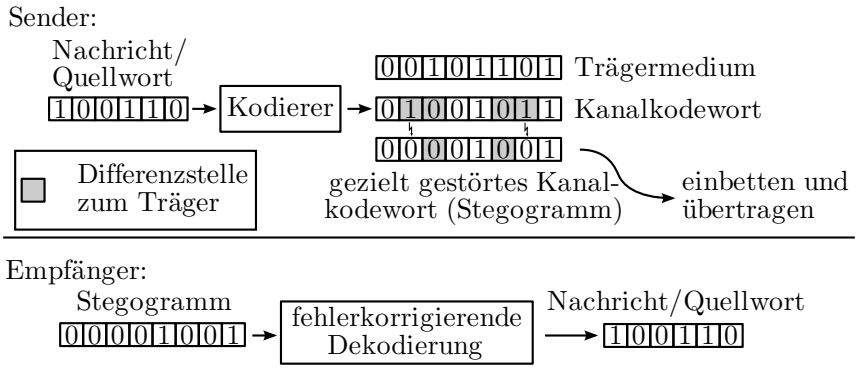


Abbildung 3.1: Ausnutzung der Fehlerkorrektur für die Steganographie

Dadurch, dass unsere Nachricht mit dem Quellkodewort gleichgesetzt wird und nicht, wie bei der klassischen Methode, die Nachricht mit dem Syndrom, berechnet sich die Einbettungsrate für so einen Kode mit

$$\alpha = \frac{l}{n}.$$

Auch die Formeln 2.1 und 2.3 für die Einbettungseffizienz bzw. die durchschnittliche Kippzahl müssen nun natürlich dahingehend angepasst werden, dass statt der redundanten Stellen k nun die Informationsstellen l des verwendeten Kodes einfließen.

Betrachten wir nun die Effizienz dieser Methode anhand eines Hammingkodes.

Beispiel 8 Gegeben sei ein Kodierer, der mit einem $(7, 4)$-Hammingkode arbeitet. Mit der vorgestellten Kodierungsmethode hat dieser Kode eine Einbettungsrate von $\alpha = \frac{4}{7} \approx 0,57$ und liegt damit Verglichen mit der Matrixkodierung zwischen dem $(3, 2)$ und dem $(7, 4)$-Hammingkode mit einer Einbettungsrate von ungefähr $0,67$ bzw. $0,43$.

Da dieser Kode Quellkodewörter der Länge 4 verarbeitet, enthält das Kanalkodealphabet 16 Kanalkodewörter. Für eine näherungsweise Berechnung von R_a nach Gleichung 2.5 sind alle Kodewörter in Tabelle 3.1 dargestellt. In der letzten Spalte der Tabelle steht die Anzahl der Bits, die geändert werden müssen, wenn das jeweilige Kodewort eingebettet wird. Diese Zahl setzt sich zum einen aus der Distanz des Kanalkodewortes zum Träger zusammen ($d_H(a, c)$), was die Distanz zum Mittelpunkt der Korrekturkugel darstellt. Davon müssen f_k Bits weniger gekippt werden, da diese

Tabelle 3.1: Kanalkodealphabet des $(7, 4)$-Hammingkodes sowie die Distanz zum Träger $c = (1101101)$

$m = a^*$	a	$a \oplus c$	Kippzahl $(d_H(a, c) - f_k)$
0000	0000000	1101101	4
0001	0000111	1101010	3
0010	0011001	1110100	3
0011	0011110	1110011	4
0100	0101010	1000111	3
0101	0101101	1000000	0
0110	0110011	1011110	4
0111	0110100	1011001	3
1000	1001011	0100110	2
1001	1001100	0100001	1
1010	1010010	0111111	5
1011	1010101	0111000	2
1100	1100001	0001100	1
1101	1100110	0001011	2
1110	1111000	0010101	2
1111	1111111	0010010	1

korrigiert werden können $(d_H(a, c) - f_k)$.[1] Für die Berechnung der mittleren Kippzahl (nach Gleichung 2.5) muss nun noch der Durchschnitt der letzten Spalte berechnet werden: $R_a \approx 2, 5$. Daraus lässt sich nun die Effizienz berechnen: $e = \frac{l}{R_a} \approx 1, 6$. Zum Vergleich: Der $(3, 2)$-Hammingkode hat mit Matrixkodierung eine Einbettungseffizienz von $e \approx 2, 67$ und der $(7, 4)$-Hammingkode eine Effizienz von $e \approx 3, 43$.

Um die mittlere Kipprate genau zu berechnen muss beachtet werden, dass in der letzten Spalte von Tabelle 3.1 in 16 von 128 Fällen für c bei der Berechnung der Kippzahl mit $d_H(a, c) - f_k$ an einer Stelle eine -1 herauskommen würde (was so natürlich nicht auftreten darf). Dieses Kodewort würde dann ohne Änderungen in der Korrekturkugel von a liegen. Um diesen Fehler noch heraus zu rechnen, müsste noch über alle Träger iteriert werden wie es in Gleichung 2.3 beschrieben ist. Die genauen Werte sind: $R_a = 2, 5078$ bzw. $e = 1, 5950$. □

Nachdem im letzten Beispiel gezeigt wurde, dass die Effizienz des Hammingkodes mit dieser Kodierungsmethode verglichen mit der klassischen Syndromkodierung nicht sehr hoch ist, könnte man argumentieren, es läge an seiner geringen Fehlerkorrektur. Genau das war ja auch eingangs die Motivation für diese Methode. Daher wollen wir

[1] Korrekt wäre hier eigentlich $d_H(a, c) - \min\{f_k; d_H(a, c)\}$, siehe Ende des Beispiels

nun am Beispiel eines Faltungskodes betrachten, wie es mit der Effizienz bei höherer Fehlerkorrektur aussieht.

Beispiel 9 Gehen wir in diesem Beispiel von einem Faltungskode mit der Generatormatrix $G = (5_8, 7_8)$ aus. Dieser hat ein Gedächtnis von 2 und eine freie Distanz von $d_f = 5$. Mit dieser Distanz kann der Dekodierer sicher 2 Fehler korrigieren. Gehen wir vereinfacht im folgenden davon aus, der Dekodierer könne genau 2 Fehler korrigieren. Wir wollen ihn mit einer Quellkodewortlänge von $l = 10$ benutzen. Die Kanalkodewortlänge ist inklusive Terminierung 24, daher ist die Einbettungsrate $\alpha = \frac{10}{24} \approx 0,42$. Er liegt damit in der Einbettungsrate sehr nahe dem $(7, 4)$-Hammingkode ($\alpha \approx 0,43$, $e \approx 3,43$).

Die durchschnittliche Kippzahl soll nun mit der Formel 2.4 ausgerechnet werden. Laut dieser Formel wählen wir uns eine Nachricht m fest und iterieren über den Träger. Die Nachricht sei das Nullwort ($m = a^* = \mathbf{0}_2$). Die Funktion $\text{code}(m, c) = \text{code}(\mathbf{0}_2, c)$ wird nun immer ein Wort ausgeben, welches bei der Dekodierung gerade noch in das Nullwort dekodiert wird. Da der Kode 2 Fehler korrigieren kann, wird das Wort allerdings immer ein Gewicht von 2 haben. Die Distanz vom Träger zu der Ausgabe der code-Funktion ist also immer gleich dem Gewicht des Trägers minus 2 ($d_H(\text{code}(\mathbf{0}_2, c), c) \approx w_H(c) - 2$).[2]

Um die mittlere Kippzahl zu berechnen muss nun noch der Durchschnitt von $w_H(c) - 2$ über alle möglichen Träger berechnet werden. Da wir alle Träger betrachten ist das durchschnittliche Gewicht des Trägers

$$\frac{1}{2^n} \sum_{c=\mathbf{0}_2}^{(2^n-1)_2} w_H(c) = \frac{|c|}{2} = \frac{n}{2} = 12.$$

Die mittlere Kippzahl des Kodes ist daher

$$R_a \approx \frac{1}{2^n} \sum_{c=\mathbf{0}_2}^{(2^n-1)_2} d_H(\text{code}(m, c), c) \approx \frac{1}{2^n} \sum_{c=\mathbf{0}_2}^{(2^n-1)_2} w_H(c) - 2 = 12 - 2 = 10.$$

Damit kann die Effizienz berechnet werden: $e = \frac{l}{R_a} \approx \frac{10}{10} = 1$.

Auch eine Abschätzung über die Effizienz größerer Kodes lässt sich so treffen. Der Faltungskode mit der Generatormatrix $G = (23_8, 35_8)$ und $l = 14$, hat eine freie Distanz von 7, eine durchschnittliche Kipprate von

$$R_a \approx \frac{1}{2^n} \sum_{c=\mathbf{0}_2}^{(2^n-1)_2} d_H(\text{code}(a^*, c), c) \approx \frac{1}{2^{36}} \sum_{c=\mathbf{0}_2}^{(2^{36}-1)_2} w_H(c) - 3 = 18 - 3 = 15$$

[2] Das gilt ähnlich wie im Beispiel 8 natürlich nur für $w_H(c) \geq 2$. Für eine approximierte Berechnung von R_a können wir diesen Fakt allerdings vernachlässigen.

und eine Effizienz von $e \approx \frac{14}{15} \approx 0,93$ bei einer Einbettungsrate von $\alpha = \frac{14}{36} \approx 0,39$.□

Wie aus den vorangegangenen Beispielen ersichtlich ist, ist diese Methode, verglichen mit der Syndromkodierung nicht sehr effizient. Das Problem besteht darin, dass die Menge der Wörter, die zu einem Quellkodewort führen, konstruktionsbedingt sehr dicht um das zum Quellkodewort gehörende Kanalkodewort herum liegen müssen.[3] Passiert es beispielsweise, dass ein kodiertes Quellkodewort zufällig dem Träger entspricht, so kann man dieses Quellkodewort mit 0 bis f_k Änderungen im Träger einbetten. Effizienter wäre es allerdings, wenn eine Änderung von einem Bit an zwei verschiedenen Stellen auch 2 verschiedene Nachrichten erzeugen würde.

Beispiel 10 Gegeben sei ein fiktiver Kode mit Quellkodewortlänge $l = 2$, Kanalkodewortlänge $n = 8$ und einem Minimalabstand $d_{min} = 7$. Es soll versucht werden die Nachricht $m = a^* = (01)$ in der Trägerfolge $c = (01110111)$ einzubetten. Das zum Quellkodewort (01) zugehörige Kanalkodewort lautet $a = (00110111)$. Eine bildhafte Darstellung zeigt Abbildung 3.2.

Hier sind links alle Quellkodewörter samt unserer gewünschten Nachricht $m = a^*$ zu sehen. Rechts sind die zu den Quellkodewörtern gehörenden Kanalkodewörter sowie deren Korrekturkugeln zu sehen. Der Träger c hat von a nur einen Abstand von 1 ($d_H(a, c) = 1$). Da unser Kode 3 Fehler korrigieren kann, kann dieser Fehler problemlos enthalten bleiben, da der Empfänger diesen wieder korrigieren kann. Dekodiert man die Trägerfolge, so ergibt sich unser gewünschtes Quellkodewort.[4]

Versuchen wir allerdings ein anderes Wort in diese Trägersequenz c einzubetten ((00), (10) oder (11)), so müssen wesentlich mehr Bits geändert werden. In Abbildung 3.2 muss der Träger c in eine andere Korrekturkugel „gezogen“ werden. Für jede Querung einer Abstandslinie muss dabei jedoch ein Bit gekippt werden.

Alle Wörter mit einem Abstand von 1 zum Träger, haben zu dem zu (01) gehörenden Kanalkodewort (a) einen Abstand von 2 oder 0. Diese Wörter wird der Kodierer auch zu (01) dekodieren, da er 2 Fehler korrigieren kann. Auf die gleiche Art kann man bei Wörtern argumentieren, die zum Träger einen Abstand von 2 haben. Um ein anderes Kodewort einzubetten muss man den Träger also an mindestens 3 Stellen stören, was zu einer schlechten Effizienz führt. □

Wie in den Beispielen 8 und 9 sowie in Abbildung 3.2 zu sehen ist, vergrößern sich die Korrekturkugeln mit der Leistungsfähigkeit eines Kodierers. Je mehr Fehler ein Kode korrigieren kann, desto größer werden die Mengen ähnlicher Wörter. Damit sinkt die Einbettungseffizienz e, wenn man die Leistungsfähigkeit eines Kodierers erhöht.

[3] genauer: Seien a und b Wörter, die zu demselben Quellkodewort führen und c ein Wort welches zu einem anderen Quellkodewort führt. Dann ist der Hammingabstand von a und b kleiner, als der von a und c ($d_H(a, b) < d_H(a, c)$).

[4] $\text{decode}(c) = \text{decode}(\text{code}(a^*) \oplus (01000000)) = m$

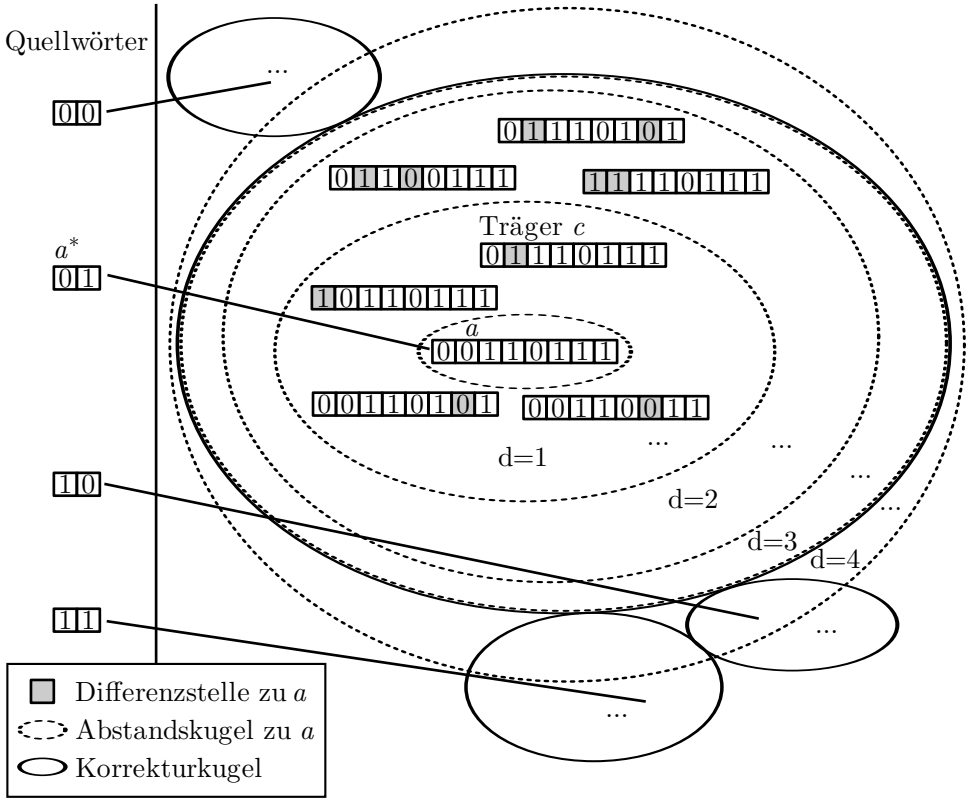


Abbildung 3.2: Kodierung mit naiver Methode. Das Quellkodewort (01) kann ohne Änderung im Träger eingebettet werden. Alle anderen Quellkodewörter könnten nur mit einem Mindestabstand von 3 eingebettet werden, was zu einer schlechten Effizienz führt

Ein weiteres Problem für die Steganographie würde sich dadurch ergeben, dass die Wahrscheinlichkeit höher ist, ein Wort anzutreffen, welches von einem Kanalkodewort einen maximalen Abstand hat, da fast immer[5] gerade so viele Bits gekippt werden, dass eine eingebettete Folge in das gewünschte Quellkodewort dekodiert wird. Wir treffen daher gehäuft auf Wörter welche sich am Rand der Korrekturkugel befinden. Das ist in Abbildung 3.3 dargestellt. Hier sind 4 Korrekturkugeln eines Kodes dargestellt. Die Wahrscheinlichkeit ein Wort anzutreffen, welches sich am Rand einer Korrekturkugel befindet ist höher, als eines zu finden, welches näher am Kanalkodewort ist. Durch diesen Umstand könnte man die Steganographie einer noch so effizienten Kodierung brechen. Ein Angriff würde so aussehen:

1. Berechne alle Kanalkodewörter des vermuteten Kodes.

[5]Nur in dem Fall, dass kein Bit gekippt werden muss, kann man auf ein anderes Wort treffen.

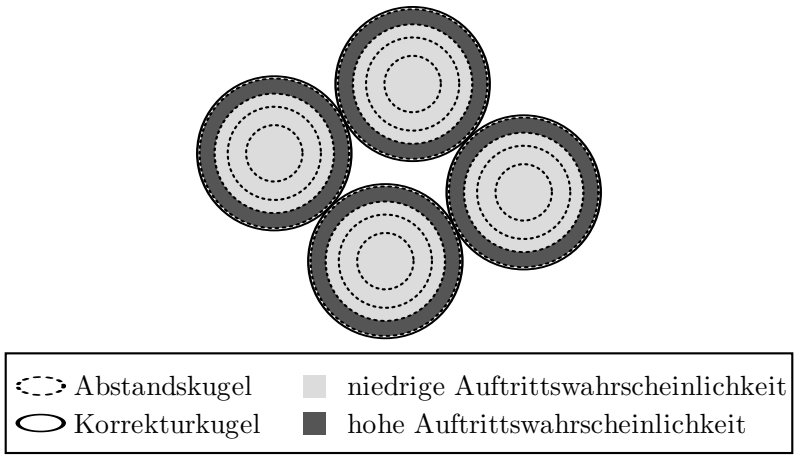


Abbildung 3.3: Angriff auf Kodierung bei Ausnutzung der Fehlerkorrektur

2. Interpretiere das Trägermedium als Kette von Wörtern.

3. Sollten gehäuft viele Wörter auftreten, die einen Abstand von f_k zu einem Kanalkodewort haben, so ist mit hoher Wahrscheinlichkeit etwas eingebettet.

Fazit dieses Abschnitts ist, dass Fehlerkorrektur nicht genutzt werden sollte, um Nachrichten für die Steganographie zu kodieren. Es ist nicht nur ineffizient, darüber hinaus kann es noch durch ein ungleiches Auftrittsverhalten von Wörtern im Steganogramm nachgewiesen werden.

3.2 Kodierung ohne festen Startzustand

Nachdem im letzten Abschnitt gezeigt wurde, dass eine Methode gefunden werden muss, in dem alle Wörter im Steganogramm gleich häufig auftreten, soll in diesem Abschnitt gezeigt werden, wie das gehen kann.

Bei jeder Kodierung findet eine Zuordnung statt, bei der jeder Nachricht und damit jedem Quellkodewort eine Menge von Wörtern zugeordnet wird. Die Wörter, die einem Quellkodewort zugeordnet sind, sollen im weiteren *Trelliswörter* genannt werden. Bei der Kodierung, wie wir sie in Abschnitt 3.1 gesehen haben, liegen die Trelliswörter, die zu einem Quellkodewort gehören dicht beieinander (alle um das Kanalkodewort herum). Das ermöglicht uns zwar eine gute Fehlerkorrektur für die Kanalkodierung, allerdings darf diese, wie im vorangegangenen Abschnitt diskutiert, nicht dazu verwendet werden, Wörter steganographisch zu kodieren, da sie nicht nur weniger effizient ist, sondern zusätzlich noch nachweisbare Spuren hinterlässt. Es muss daher nach einer Methode gesucht werden, diese Trelliswörter anders zu erzeugen als auf die herkömmliche Kodierungsart. Hierbei dürfen Trelliswörter die

einem Quellkodewort zugeordnet sind nicht mehr dicht beieinander liegen, was die Fehlerkorrektureigenschaft des Kodes eliminiert.

Eine Möglichkeit, die die Trelliswörter auf andere Weise den Quellkodewörtern zuordnet und damit die fehlerkorrigierende Eigenschaft entfernt, ist auf die Festlegung des Startzustandes sowie Terminierung zu verzichten. Diese ist in Abbildung 3.4 dargestellt. Sie ist der Methode, die von Miller et al. vorgestellt wurde ähnlich.[6] Dort

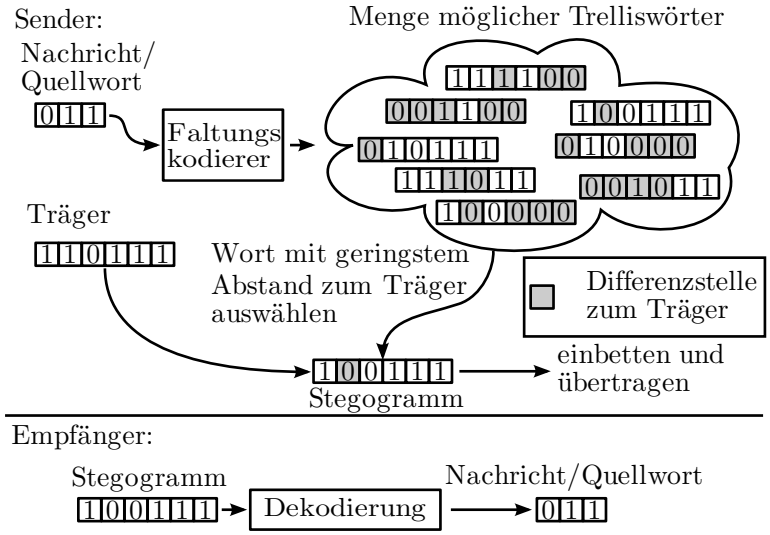


Abbildung 3.4: Beispiel für eine Kodierung mit variablen Startzustand. (Generatormatrix $G = (15_8, 17_8)$; Quellkodewort $a^* = (011)$; Träger $c = (110111)$)

wurden jedoch Trelliskodes mit zufälliger Kantenbeschriftung und anderer Folgezustandsabbildung gewählt. Ein genauerer Vergleich befindet sich in Abschnitt 3.5.

Das Quellkodewort wird vom Faltungskodierer mit jedem Zustand als Startbelegung einmal kodiert. Aus dieser Menge von resultierenden Trelliswörtern kann das mit dem kleinsten Abstand zum Trägermedium gewählt werden. Der Empfänger kann das Quellkodewort wiederfinden, indem er ein komplettes Trellisdiagramm aufbaut und den Pfad auswählt, der mit dem Steganogramm übereinstimmt.[7]

Um Doppeldeutigkeiten bei der Dekodierung zu vermeiden, ist zu beachten, dass das Quellkodewort eine bestimmte Mindestlänge hat. Dazu ein Beispiel:

Beispiel 11 Betrachten wir einen Faltungskode mit 8 Zuständen ($8 = 2^{\hat{m}} \rightarrow \hat{m} = 3$), einer Koderate von $R = \frac{1}{2}$ und wir wollen 2 Bit einbetten ($|a^*| = l = 2$). Aufgrund

[6] M.L. Miller, G.J. Doërr und I.J. Cox, Applying informed coding and embedding to design a robust high-capacity watermark. Image Processing, IEEE Transactions on, 13 Juni 2004:6.

[7] Natürlich kann an dieser Stelle auch der Viterbi-Algorithmus verwendet werden und der Pfad mit der Distanz gleich Null ausgewählt werden.

der Koderate von $\frac{1}{2}$ gibt der Kodierer bei 2 Bit Input 4 Bit aus ($\frac{l}{R}$). Eine Ausgabe aller Trelliswörter für einen Kodierer mit der Generatormatrix $G = (15_8, 17_8)$ ist in Tabelle 3.2 dargestellt.

Tabelle 3.2: Doppeldeutigkeiten bei kurzem Quellkodewort. (Generatormatrix $G = (15_8, 17_8)$)

		Startzustand							
		000	001	010	011	100	101	110	111
Quellwort	00	0000	1100	0111	1011	1101	0001	1010	0110
	01	0011	1111	0100	1000	1110	0010	1001	0101
	10	1111	0011	1000	0100	0010	1110	0101	1001
	11	1100	0000	1011	0111	0001	1101	0110	1010

Betrachtet man alle Möglichkeiten, ein Kodewort zu bilden, so kann man von jedem der 8 Zustände in jedem Schritt 2 Pfade gehen, je nachdem ob in diesem Schritt 0 oder 1 eingebettet wurde. Da wir 2 Bit einbetten, können wir auf diese Weise $8 \cdot 2^2 = 2^{\hat{m}} \cdot 2^l = 32$ Trelliswörter erzeugen. Da jedes dieser 32 Trelliswörter nur 4 Bit lang ist und sich mit 4 Bit insgesamt nur 16 Wörter darstellen lassen ($2^{\frac{l}{R}}$), muss es Wörter geben, die mehr als einmal vorkommen. Dadurch kann es zu Doppeldeutigkeiten bei der Dekodierung kommen.

Es besteht zwar die Möglichkeit, dass ein Kode alle Doppeldeutigkeiten in das gleiche Kodewort dekodiert,[8] allerdings soll hier eine solche nicht betrachtet werden.□

Als zwingende Grenze für eine eindeutige Dekodierung muss daher die Ungleichung $2^{\frac{l}{R}} \geq 2^{\hat{m}} \cdot 2^l$ bzw.

$$l \geq \frac{\hat{m}}{\frac{1}{R} - 1}$$

eingehalten werden.

Neben einer unteren Schranke für l stellt sich die Frage auch, ob man eine obere Schranke angeben kann, für die eine Kodierung sinnvoll ist. Da der Faltungskodierer das Quellkodewort nacheinander in sein Gedächtnis schreibt, erreicht er nach $\hat{m}$ Schritten einen Zustand, welcher nur noch von der einzubettenden Folge abhängt, nicht mehr vom Startzustand. Daher ist auch die Ausgabe ab diesem Zeitpunkt von allen Startzuständen aus gesehen gleich. Ein Beispiel dafür ist in Tabelle 3.3 dargestellt. Hier wurde ein Quellkodewort der Länge 6 mit einem Faltungskodierer mit einem Gedächtnis von 3 kodiert. Zu sehen ist, dass die Trelliswörter sich nach dem dritten Ausgabeschritt nicht mehr unterscheiden.

[8] In Tabelle 3.2 müssten dann alle doppeldeutigen Wörter paarweise in der gleichen Zeile liegen.

Tabelle 3.3: Beispielkodierung eines unnötig langen Wortes
(Generatormatrix $G = (15_8, 17_8)$; Quellkodewort $a^* = (001\,010)$)

Startzustand	Trelliswort
000	00 00 11 11 10 00
001	11 00 11 11 10 00
010	01 11 11 11 10 00
011	10 11 11 11 10 00
100	11 01 00 11 10 00
101	00 01 00 11 10 00
110	10 10 00 11 10 00
111	01 10 00 11 10 00

Daher kann ab diesem Zeitpunkt auch keine Effizienzsteigerung mehr stattfinden indem man den Startzustand variiert. Die obere Schranke für die Quellkodewortlänge ist das Gedächtnis des Kodierers. Es gilt also:

$$l \leq \hat{m}.$$

Fasst man beide Ungleichungen zusammen, so erhält man einen Bereich, in dem mit der Quellkodewortlänge experimentiert werden kann:

$$\frac{\hat{m}}{\frac{1}{R} - 1} \leq l \leq \hat{m}. \tag{3.1}$$

Da für Kodes mit der Koderate $\frac{1}{2}$ die untere Schranke gleich der oberen Schranke ist ($\frac{\hat{m}}{\frac{1}{R}-1} = \hat{m}$), kann hier $\hat{m} = l$ geschlussfolgert werden. Auch für Kodes mit kleinerer Koderate ist es besser, die untere Schranke mit Gleichheit zu erfüllen. In Tabelle 3.3 ist schon zu sehen, dass die Hälfte aller Zustände nach 2 Schritten den gleichen Output erzeugt und nach dem ersten Schritt je 2 Zeilen den gleichen Output beinhalten. Auch das kommt natürlich daher, dass in jedem Schritt das letzte Bit des Zustandes wegfällt[9] und ein neues vom Quellkodewort eingefügt wird. Daher wird beispielsweise bei Eingabe einer 0 aus den Startzuständen 100_2 und 101_2 der Zustand 010_2. Beide produzieren ab diesem Zeitpunkt den gleichen Output. Daher ist es besser, kürzere Wörter einzubetten, um mehr Auswahl in den Zuständen zu haben, am besten, die untere Schranke immer mit Gleichheit zu erfüllen.

Bei der Auswertung stellte sich heraus, dass die Einbettungseffizienz mit Erhöhung des Gedächtnisses kontinuierlich stieg. Die höchste Effizienz ($e \approx 3,765$) wurde bei einem Faltungskode mit der Generatormatrix $(40511_8, 51565_8, 60075_8)$ erzielt. Weitere Ergebnisse sind in Abbildung 3.5 dargestellt.

[9] es wird aus dem Schieberegister herausgeschoben

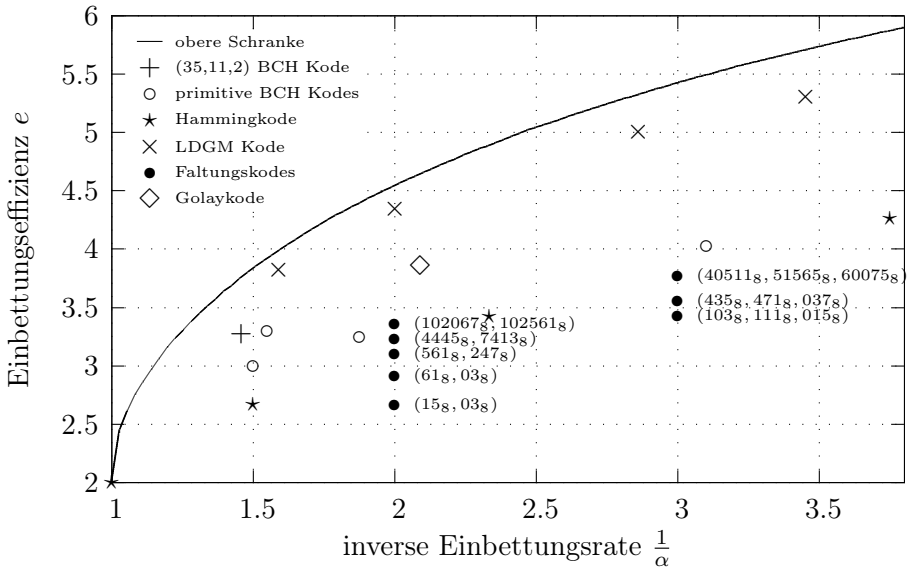


Abbildung 3.5: Einbettungseffizienz verschiedener Faltungskodes mit variablem Startzustand

Natürlich steigt die Berechnungskomplexität mit dem Gedächtnis. Mit Hinzunahme eines Speicherelements wird die Anzahl der Zustände des Trellisdiagramms verdoppelt. Da die Kodierung darauf beruht, von jedem Zustand einmal eine Kodierung durchzuführen, steigt der Aufwand somit exponentiell mit der Anzahl der Speicherelemente. Bei der Dekodierung steigt der Aufwand ähnlich. Hier muss zuerst ein verkürztes Trellisdiagramm erstellt werden. Aus diesem müssen im ersten Schritt alle Kanten gesucht werden, die den ersten $\frac{1}{R}$ Bits gleichen. Anschließend muss vom Zielzustand dieser Kanten der richtige Pfad weiter gesucht werden. In jedem Schritt verringert sich so die Zahl der zu durchsuchenden Kanten. Generell besteht hier aber der gleiche exponentielle Aufwand bezogen auf die Anzahl der Speicherelemente.

Wird die untere Schranke mit Gleichheit erfüllt, so gilt $\hat{m} \geq l$. Aus diesem Zusammenhang kann eine weitere Optimierung des Dekodieralgorithmus vorgenommen werden. Nachdem im Dekodierprozess das Trellisdiagramm aufgebaut wurde und der zum Steganogramm zugehörige Pfad lokalisiert wurde, braucht dieser nicht zurückverfolgt werden um die Nachricht zu bestimmen. Es reicht, die Bits aus dem Endzustand auszulesen, der nun das Quellkodewort enthält.

Beispiel 12 Gegeben sei ein Faltungskodierer mit einem Gedächtnis von 8 und einer Koderate von $\frac{1}{3}$. Um die untere Schranke mit Gleichheit zu erfüllen, würden wir versuchen, Quellkodewörter der Länge 4 einzubetten. Endet im Trellisdiagramm der zum Steganogramm zugehörige Pfad im Zustand (10110101), so handelt es sich

bei den ersten 4 Bit (1011) um unser gesuchtes Quellkodewort und damit um die Nachricht. Der Pfad braucht nicht mehr rückwärts ausgelesen werden. □

3.3 Verbesserung der Einbettungsrate durch Einbeziehung des Startzustandes in die Kodierung

In Tabelle 3.4 ist der Kodierungsprozess aus Abbildung 3.4 nochmal tabellarisch dargestellt. Auffällig ist dabei, dass es egal ist, ob das Quellkodewort mit (100111) oder mit (010111) kodiert wird. In beiden Fällen muss ein Bit vom Träger gekippt werden. Interessant ist daher die Frage, ob sich diese Mehrdeutigkeit, für die Kodierung ausnutzen lässt, um die Effizienz oder die Einbettungsrate zu steigern.

Tabelle 3.4: Beispielkodierung für ein Quellkodewort mit variablem Startzustand. (Generatormatrix $G = (15_8, 17_8)$; Quellkodewort $a^* = (011)$; Träger $c = (110111)$)

Startzustand	Trelliswort (a)	Abstand zum Träger ($w_H(a \oplus c)$)
000	001100	5
001	111100	3
010	010000	4
011	100000	4
100	111011	2
101	001011	4
110	100111	1
111	010111	1

Esen et al.[10] stellte schon eine Möglichkeit vor, wie Startzustände in die Kodierung von Wasserzeichen mit aufgenommen werden können. Diese Methode wollen wir hier für die Steganographie nutzen.

In unserem einführenden Beispiel (Tabelle 3.4) hätte man das erste Bit des Startzustandes benutzen können um zusätzlich eine 0 oder 1 einzubetten. So kann man pro Einbettungsschritt 4 statt 3 Bit einbetten, hat also eine Quellkodewortlänge von 4. Das vierte Bit wählt aus, ob wir das tatsächlich eingebettete Wort aus der oberen Hälfte von Tabelle 3.4 nehmen oder aus der unteren.

Der Empfänger dekodiert wie gewohnt, nur dass er zusätzlich darauf achtet, welchen Startzustand der Pfad des empfangenen Trelliswortes hat. Das erste Bit des Startzustandes fügt er anschließend dem dekodierten Quellkodewort wieder hinzu. Da wir nicht mehr 3 sondern 4 Bit Quellkodewort auf 6 Bit im Trägermedium einbetten,

[10] E. Esen, A.A. Alatan und M. Askar, Trellis coded quantization for data hiding. in: EUROCON 2003. Computer as a Tool. The IEEE Region 8. Band 2, September 2003.

erhöht sich die Einbettungsrate von $\frac{3}{6}$ auf $\frac{4}{6}$ bzw. von $\frac{1}{2}$ auf $\frac{2}{3}$. Natürlich ist es möglich mehrere Bits statt nur einem zu verwenden, was die Einbettungsrate weiter erhöht.

Sei l_Z die Anzahl der Bits, die zusätzlich, zur normalen Kodierung vom Zustand benutzt werden und l die Länge des Quellkodewortes, welches der Faltungskodierer im normalen Modus kodiert, dann berechnet sich die Einbettungsrate allgemein durch:

$$\alpha = R + \frac{l_Z}{n} = \frac{l + l_Z}{n}$$

Betrachten wir jedoch die Effizienz: Wäre das zusätzlich einzubettende Bit eine 0 (nur die obere Hälfte ist für einen Startzustand zulässig), so hätte man statt eines Bits gleich drei kippen müssen (Startzustand 001 mit $w_H((111100) \oplus c) = 3$). Nimmt man jedoch statt des ersten Bits des Startzustandes das letzte und wählt damit aus, ob gerade oder ungerade Zustände als Startzustand zulässig sind, ist es egal, ob man eine 0 oder eine 1 einbetten will. Da sowohl vom Startzustand 110 als auch vom Startzustand 111 der Abstand zum Träger 1 ist ($w_H(a \oplus c) = 1$), ist in beiden Fällen 1 Bit im Trägermedium zu kippen.

Sehr offensichtlich ist, dass mit Erhöhung der Einbettungsrate die Menge der Trelliswörter die zur Auswahl stehen verringert wird und damit die mittlere Kippzahl steigt.[11] Das führt jedoch nicht unbedingt unmittelbar zu einer Senkung der Einbettungseffizienz, da hierfür auch die gesamte Nachrichtenlänge direkt mit eingeht, welche mit Erhöhung von l_Z steigt ($e = \frac{l+l_Z}{R_a}$).

Die scheinbar zufällige Beobachtung, dass es besser ist, als zusätzliche Kodierung die hinteren Bits zu benutzen, bestätigt sich in der empirischen Untersuchung. Eine Erklärung für diese Beobachtung ist, dass Zustände, die sich in den vorderen Bits nicht unterscheiden (z. B. 010 und 011), ähnliche Trelliswörter erzeugen.

Auch in Tabelle 3.4 ist dieser Effekt zu sehen. Vergleicht man dort immer 2 benachbarte Zeilen (und damit Zustandspaare, die sich nur im letzten Bit unterscheiden) so fällt auf, dass die Trelliswörter in den letzten 4 Bit gleich sind. Betrachtet man 4 benachbarte Zeilen (also obere bzw. untere Hälfte der Tabelle) so sind die letzten 2 Bit aller 4 Trelliswörter gleich.

Anders ist das, wenn wir alle ungeraden Zeilen betrachten bzw. je zwei Zeilen vergleichen, in denen sich der Startzustand in den hinteren beiden Bits nicht unterscheidet. In diesen Fällen gibt es weniger Gleichheiten zwischen den Trelliswörtern. Benutzt man für die Kodierung der zusätzlichen Bits die hinteren Bits des Zustandes, so erreicht man genau diesen Effekt, dass man aus einer Menge von Wörtern auswählen kann, in denen die Trelliswörter alle eher unterschiedlich aussehen. Damit steigt auch die Wahrscheinlichkeit, dass ein Kodewort enthalten ist, welches dem Träger ähnlich ist. Eine grafische Darstellung der Methode ist in Abbildung 3.6 gegeben.

[11] Schränkt man die Startzustände in Tabelle 3.4 weiter ein ($l_Z = 2$), so erhöht sich auch hier die mittlere Kippzahl. Allgemein: $l_Z \leq l'_Z \implies R_a^{l_Z} \leq R_a^{l'_z}$

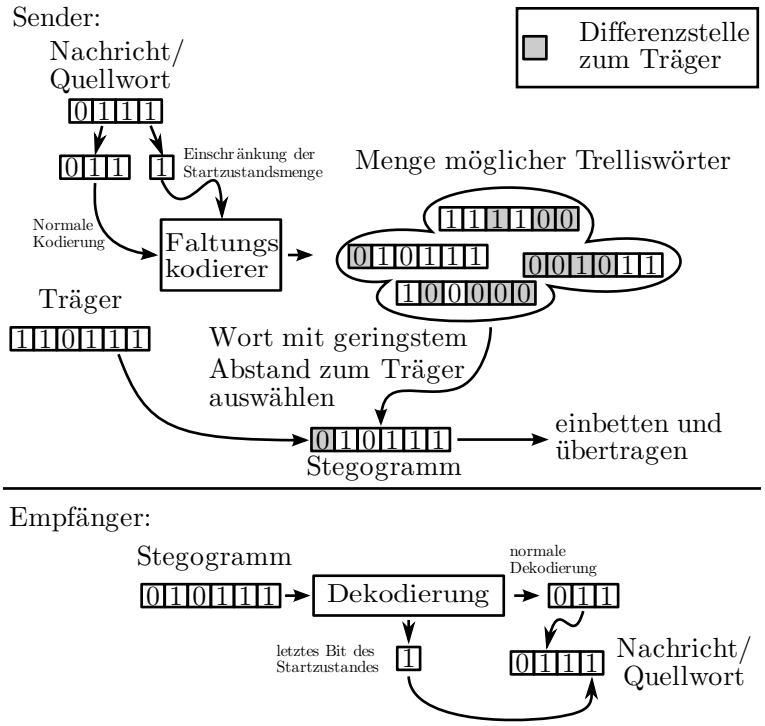


Abbildung 3.6: Beispiel für Kodierung des Startzustandes

Eine Darstellung über die Effizienz dieses Einbettungsvorgehens befindet sich in Abbildung 3.7. Die Punkte bei der inversen Einbettungsrate von 2 sind die Kodes, die auch schon in Abbildung 3.5 zu sehen waren. Genau genommen stellt das Verfahren aus Abschnitt 3.3 einen Spezialfall dieses Verfahrens dar, in dem die Anzahl der zusätzlichen Bits 0 ist ($l_Z = 0$). Die zu diesen Punkten zugehörigen Punkte sind mit einer Linie verbunden. Hierbei ist der nächst linke Punkt derjenige, bei dem ein Bit aus den Zuständen verwendet wurde ($l_Z = 1$), der darauf folgende steht für $l_Z = 2$ usw.. Jede Linie endet im Ursprung. Hier werden alle Bits des Startzustandes zur Kodierung verwendet. Man fügt keinerlei Redundanz mehr hinzu, die Wörter werden nur noch umsortiert.

Auffallend ist, dass die Effizienz bei Hinzunahme eines Zustandes ($l_Z = 1$) in den meisten Fällen nahezu der Effizienz gleicht, die derselbe Kode ohne Benutzung eines Zustandes hat ($l_Z = 0$).

In Abbildung 3.8 ist die Effizienz von Faltungskodes der Koderate $\frac{1}{3}$ denen mit einer Koderate von $\frac{1}{2}$ gegenübergestellt. Dabei ist auffällig, dass die Kodes mit einer Koderate von $\frac{1}{3}$ schlechter sind, als Kodes der Koderate $\frac{1}{2}$, bei gleicher Einbettungsrate.

Beispiel 13 Betrachten wir den Kode mit der Generatormatrix $(435_8, 471_8, 037_8)$. Dieser hat durch seine 3 Ausgänge eine Koderate von $\frac{1}{3}$. Die optimale Quellkode-

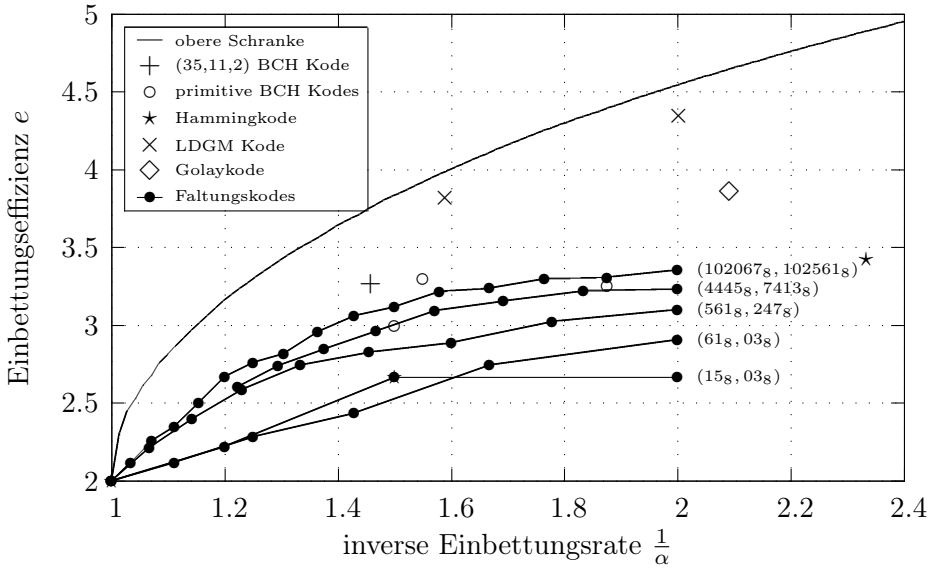


Abbildung 3.7: Einbettungseffizienz verschiedener Faltungskodes der Koderate $\frac{1}{2}$ mit Zustandskodierung

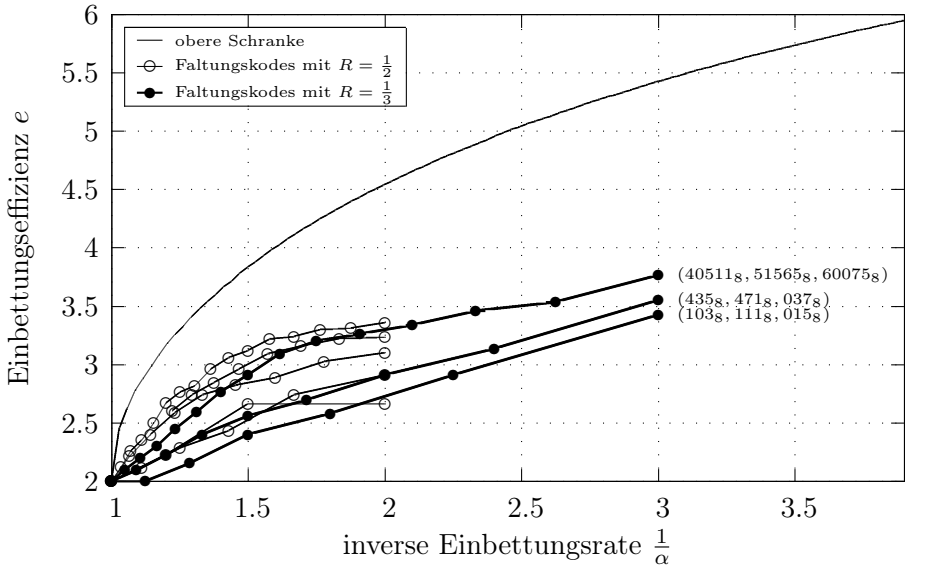


Abbildung 3.8: Vergleich der Einbettungseffizienz verschiedener Faltungskodes der Koderate $\frac{1}{2}$ und $\frac{1}{3}$ mit Zustandskodierung

wortlänge für diesen Kode ergibt sich, wenn er die untere Schranke mit Gleichheit erfüllt und ist damit 4.[12] Die Trelliswortlänge ist in diesem Fall 12. Nimmt man noch 2 Bit für die Kodierung durch die Zustände, so erreicht man eine Einbettungsrate von $\frac{6}{12} = \frac{1}{2}$. Bei dieser Koderate ist der Kode allerdings weniger effizient als ein vergleichbarer Kode der Koderate $\frac{1}{2}$ mit einem Gedächtnis von 8. Des Weiteren ist die Berechnung komplexer als die Berechnung, die für einen Faltungskode mit 2 Ausgängen nötig ist. Es ist in diesem Fall also günstiger einen Kode mit Koderate $\frac{1}{2}$ ohne zusätzliche Kodierung von Zuständen zu verwenden. □

Angemerkt sei allerdings noch, dass es sich dabei nur um die Beobachtungen handelt, die während der Versuche gemacht wurden, was nicht ausschließt, dass es Kodes mit einer Ausgangskoderate von $\frac{1}{3}$ gibt, bei denen die Effizienz hier höher ist.

3.4 Rekursive Faltungskodierer

Die Herangehensweise für die Benutzung rekursiver Faltungskodes ist dieselbe wie in Abschnitt 3.2 bereits vorgestellt. Im Unterschied zur Kodierungstheorie, wo systematisch rekursive Faltungskodes benutzt werden, sind diese hier allerdings völlig ungeeignet. Würde man einen Ausgang direkt mit dem Eingang verbinden, so würde das für dieses Bit schon einer normalen LSB-Steganographie gleichen. Hier würde man eine einfache LSB-Steganographie immer vorziehen.

Für die Steganographie lassen sich rekursive Faltungskodierer dennoch nutzen, indem man den bisher untersuchten Kodierern eine Rückkopplung anfügt. In den durchgeführten Versuchen konnte zu jedem Kodierer ein mit Rekursion effizienterer Kodierer gefunden werden. Dies ist in Abbildung 3.9 zu sehen.

Zu beachten ist hierbei, dass die Rekursion, anders als im klassischen Fall, keinen unmittelbaren Einfluss auf die Ausgabe hat. In Abbildung 3.10 ist ein Schaltbild für einen rekursiven Faltungskodierer, wie er hier verwendet wurde, nochmals dargestellt. Zu sehen ist, dass die Rekursion nicht vor dem ersten Knotenpunkt addiert wird, sondern erst danach. Das hat zur Folge, dass die Kantenbeschriftung (Ausgabevektoren) im Trellisdiagramm für diesen Kode genauso aussehen, wie die für einen Kode gleicher Generatormatrix, aber ohne Rekursion. Der Unterschied liegt im Folgezustand. Solch ein Vergleich ist in Abbildung 3.11 in Form eines verkürzten Trellisdiagramms dargestellt.

Auffallend ist, dass in der nicht rekursiven Variante jeder Zustand der mit einer 0 beginnt auch mit einer Eingabe 0 erreicht wird, da das Eingangsbit direkt auf das erste Speicherelement geschoben wird. Das wird durch die Rekursion aufgehoben. Hier führt immer eine durchgezogene sowie eine gestrichelte Kante zu einem Folgezustand.

[12] $\frac{\hat{m}}{\frac{1}{R}-1} = \frac{8}{3-1} = 4$ vgl. Formel 3.1

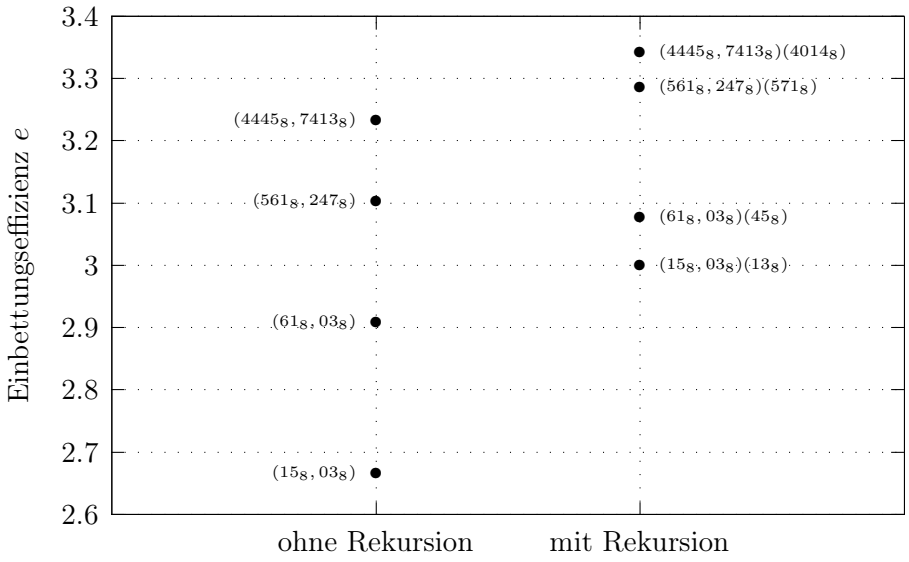


Abbildung 3.9: Vergleich der Einbettungseffizienz von Kodes ohne Rekursion und Kodes mit Rekursion

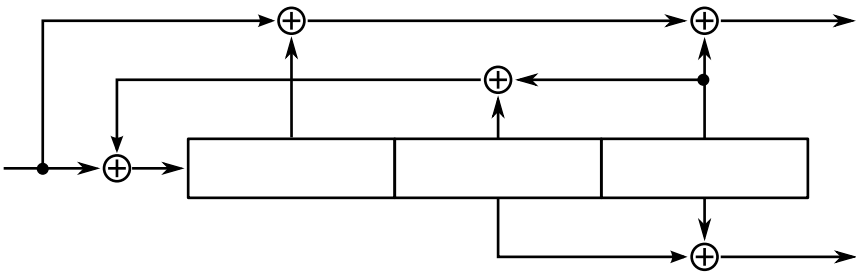


Abbildung 3.10: Schaltbild für den rekursiven Faltungskodierer mit der Generatormatrix $G = (15_8, 03_8)$ und einer Rekursion von (13_8)

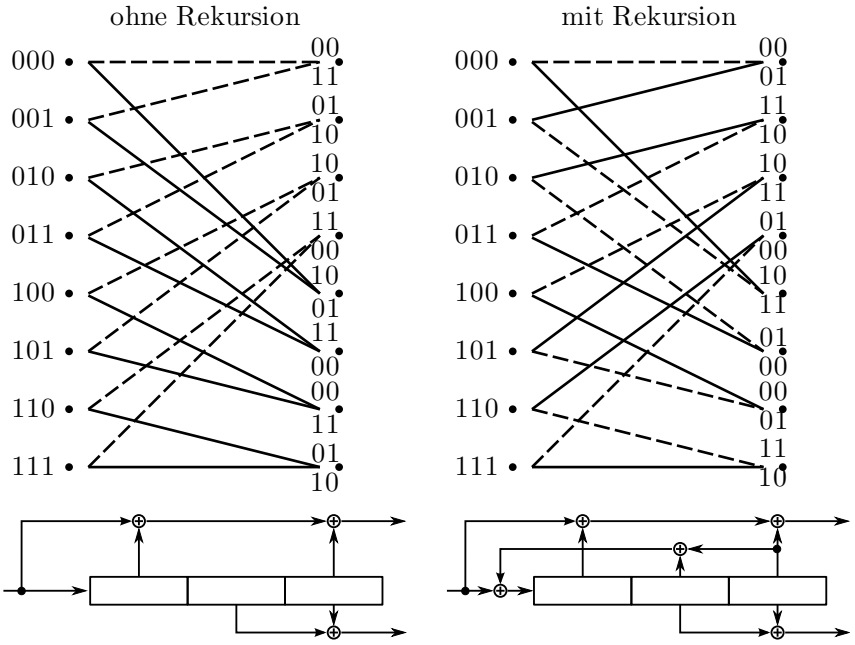


Abbildung 3.11: Vergleich der Trellisdiagramme zweier Faltungskodierer mit der Generatormatrix $G = (15_8, 03_8)$. Links ohne Rekursion und rechts mit einer Rekursion von (13_8)

Des Weiteren fällt auf, dass mit Rekursion die Eigenschaft verloren geht, dass komplementäre Ausgabesequenzen[13] auftreten. Auffällig bei der Untersuchung nichtrekursiver Faltungskodierer war, dass sie immer dann gut für die Steganographie geeignet waren, wenn sie keine komplementären Ausgänge besitzen und diese Faltungskodierer mittels hinzufügen einer Rekursion immer dann verbessert werden konnten, wenn die komplementären Eingänge verschwanden.

Diese zwei Beobachtungen deuten darauf hin, dass Faltungskodes besser für die Steganographie geeignet sind, wenn sie eine weniger gleichmäßige Struktur besitzen.

Die Optimierung, die in Abschnitt 3.2 (vgl. auch Beispiel 12) vorgestellt wurde, dass das Wort direkt aus dem Endzustand abgelesen werden kann, kann hier so nicht mehr durchgeführt werden, da die Nachricht durch die Rekursion nicht mehr ohne Veränderung in das Schieberegister geschoben wird.

[13] vgl. dazu Abschnitt 2.2.1

3.5 Vergleich zu Miller et al.

Miller et al. zeigten, wie sich Trelliskodes nutzen lassen, um robuste Wasserzeichen zu erstellen.[14] Dafür benutzen sie Trelliskodes mit zufälliger Kantenbeschriftung. Im klassischen Fall hat jeder Zustand 2 Ausgänge. Mit einem wird die 0 kodiert mit dem anderen die 1. Miller lässt in seinen Trellisdiagrammen auch mehr als 2 Zustandsübergänge zu. Ein Beispiel für ein verkürztes Trellisdiagramm mit 4 Zuständen und 4 Zustandsübergängen pro Zustand ist in Abbildung 3.12 dargestellt.

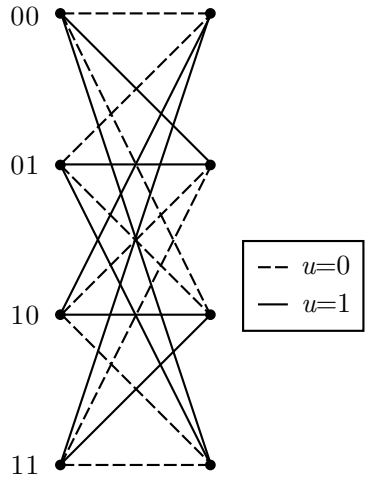


Abbildung 3.12: Verkürztes Trellisdiagramm mit 4 Zuständen und 4 Zustandsübergängen je Zustand

Genauso, wie die Kodierung in Abschnitt 3.2 vorgestellt wurde, verwendet auch Miller keinen festen Startzustand, sowie keine Terminierung. Für die Dekodierung schlägt Miller den Viterbi-Algorithmus vor, was in diesem Fall sinnvoll ist, da bei einem Wasserzeichen davon ausgegangen werden muss, dass es gestört wurde. Für die Steganographie reicht es nach dem Pfad zu suchen, der mit der Empfangsfolge übereinstimmt, was weniger Berechnungsaufwand verursacht. Wie auch bei Rekursiven Faltungskodierern kann hier die Optimierung, aus Abschnitt 3.4 (vgl. auch Beispiel 12), nicht angewandt werden, da das Trellisdiagramm nicht aus einem Schieberegister erzeugt wurde und somit der Endzustand unabhängig vom Quellkodewort ist.

Ein Problem ergibt sich dadurch, dass die Kantenbeschriftung und damit die Ausgabe zufällig gewählt ist. Um der Anforderung der Steganographie gerecht zu werden und unerkannt zu bleiben müssen alle möglichen Wörter der Länge n gleich oft als Trelliswort ausgewählt werden. Das geht natürlich nur, wenn die Wörter mit denen die Kanten des Trellisdiagramms beschriftet sind gleich häufig auftreten.

[14] M.L. Miller, G.J. Doërr und I.J. Cox, Dirty-paper trellis codes for watermarking. in: Image Processing. 2002. Proceedings. 2002 International Conference on. Band 2, September 2002.

Da sich die Kantenbeschriftung eines Trellisdiagramms bei Faltungskodes aus einer Linearkombination der Speicherelemente zusammensetzt ist eine Gleichverteilung hier gegeben.

Zusätzlich besteht das Problem, dass die Kodewörter eindeutig dekodiert werden müssen. Ist die Kantenbeschriftung lang genug und jede Kante des verkürzten Trellisdiagramms hat eine andere Beschriftung, so kann in jedem Schritt ein eindeutiger Pfad bestimmt werden. Eine so lange Kantenbeschriftung ist jedoch für die Steganographie unpraktikabel, da die Koderate aus der Länge der Kantenbeschriftung berechnet wird.

Beispiel 14 Angenommen sei ein Kodierer mit 128 Zuständen. Jeder Zustand hat 4 Folgezustände. Insgesamt hätte das Trellisdiagramm $128 \cdot 4 = 512$ Pfade. Um jedem Pfad eine eindeutige Beschriftung zuzuweisen, müssten sie mit 9 Bit beschriftet sein. Dadurch hätte der Kodierer nur noch eine Koderate von $\frac{1}{9}$. □

Versuchen wir also bei einer Koderate zwischen $\frac{1}{2}$ und $\frac{1}{3}$ etwas einzubetten, so dürfen für eine garantiert eindeutige Kodierung maximal 8 Pfade im verkürzten Trellisdiagramm vorhanden sein. Da diese Maßgabe für das Ziel einer guten Einbettungseffizienz nicht einzuhalten ist, müssen weitere Bedingungen gefunden werden die nötig sind um eine eindeutige Dekodierung zu gewährleisten.[15] Eine solche Suche sollte wahrscheinlich in die Richtung gehen, dass die Wörter aller Zyklen im Zustandsautomaten paarweise verschieden sein müssen.

3.6 Trellis+1, Trellis+2

Wie schon in Abschnitt 2.4 beschrieben, wurde von Zhang et. al. gezeigt, wie die Einbettungseffizienz von Blockkodes verbessert werden kann. Dieses Schema lässt sich problemlos auf Faltungskodes übertragen, da wir uns vor dem Einbetten auf eine Blockgröße festlegen. Wieder haben wir die 2 Gleichungen zu erfüllen:

$$(m_1, \ldots, m_l) = \text{decode}(b(x_1), \ldots, b(x_n)) \tag{3.2}$$

$$m_{l+1} = \left(\left\lfloor \frac{x_1}{2} \right\rfloor + \ldots + \left\lfloor \frac{x_n}{2} \right\rfloor + x_{n+1} \right) \bmod 2 \tag{3.3}$$

Um die mittlere Kippzahl zu messen müsste nun wiederum eine Testreihe nach Gleichung 2.3 aufgestellt werden. Es kann allerdings auch einfacher vorgegangen werden um die mittlere Kippzahl eines Trellis+1 (R_a^{+1}) anhand der mittleren Kippzahl des unmodifizierten Schemas abzuschätzen. Dazu benutzen wir eine abgewandelte Form von Gleichung 2.5 und iterieren für die Berechnung über alle möglichen Nachrichten (bzw. Quellkodewörter). Es werden 2 Fälle unterschieden:

[15] Um ausreichend genaue Bedingungen zu finden reichte der Zeitrahmen dieser Arbeit leider nicht aus, vgl. Abschnitt 4.2

1. Um Gleichung 3.2 zu erfüllen muss kein Wert von x_1 bis x_n geändert werden. Dieser Fall tritt höchstens für genau eine Belegung von $(b(x_1), \ldots, b(x_n))$, also insgesamt zweimal auf. Um Gleichung 3.3 zu erfüllen, muss x_{n+1} mit einer Wahrscheinlichkeit von $0,5$ angepasst werden.

2. Um Gleichung 3.2 zu erfüllen muss mindestens ein Wert von x_1 bis x_n geändert werden. Dieser Fall tritt $2^{l+1} - 2$ mal auf. Um Gleichung 3.3 zu erfüllen muss x_{n+1} nicht verändert werden.

Zusammengefasst lässt sich folgende Formel aufstellen:

$$R_a^{+1} \leq \frac{(0,5 + R_a) \cdot 2 + R_a \cdot (2^{l+1} - 2)}{2^{l+1}} = \frac{R_a \cdot 2^{l+1} + 1}{2^{l+1}}.$$

Tritt Fall 1 genau zweimal auf, so wird die Formel mit Gleichheit erfüllt.

Erhöht man die Anzahl zusätzlich eingebetteter Bits auf 2 so treten für die Berechnung von R_a^{+2} 4 Fälle auf:

1. Muss x_1 bis x_n nicht geändert werden, so müssen mit einer Wahrscheinlichkeit von $\frac{1}{4}$ entweder sowohl x_{n+1} als auch x_{n+2} geändert werden oder mit einer Wahrscheinlichkeit von $\frac{1}{2}$ eines von beiden. Iteriert man über alle möglichen Wörter $(b(x_1) \ldots b(x_n))$, so tritt dieser Fall maximal einmal auf.

2. Wird nur ein Wert von x_1 bis x_n geändert, so muss mit einer Wahrscheinlichkeit von $\frac{1}{2}$ x_{n+1} oder x_{n+2} geändert werden, was für alle $(b(x_1) \ldots b(x_n))$ in n Fällen auftreten kann.

3. Werden von x_1 bis x_n 2 Werte geändert, so lassen sich mit $\left(\left\lfloor \frac{x_1}{2} \right\rfloor + \left\lfloor \frac{x_2}{2} \right\rfloor\right) \mod 4$ $(x_i \in \{0, 1, 2\})$ nur 3 Werte darstellen. Es muss also mit einer Wahrscheinlichkeit von $\frac{1}{4}$ ein zusätzlicher Wert von x_{n+1} oder x_{n+2} gekippt werden. Das kann für $(b(x_1) \ldots b(x_n))$ in maximal $\binom{n}{2}$ von Fällen auftreten.

4. In allen Fällen in denen von x_1 bis x_n 3 Werte geändert werden, muss kein zusätzliches Bit für m_{l+1} geändert werden. Mit $\left(\left\lfloor \frac{x_1}{2} \right\rfloor + \left\lfloor \frac{x_2}{2} \right\rfloor + \left\lfloor \frac{x_3}{2} \right\rfloor\right) \mod 4$ $(x_i \in \{0, 1, 2\})$ lassen sich alle Werte in Z_4 darstellen. Dieser Fall tritt immer dann auf, wenn keiner der vorangegangenen Fälle aufgetreten ist.

In diesen Fällen wurde bisher immer das Maximum für alle möglichen Wörter der Form $(b(x_1) \ldots b(x_n))$ betrachtet. Jeder dieser Fälle kann allerdings einmal für jede Kombination aus $(m_{l+1} m_{l+2})$ auftreten. Es muss daher wenn alle 2^{l+2} Nachrichten betrachtet werden die durchschnittlichen Änderungen wie sie bisher in den Fällen diskutiert wurden mit 4 multipliziert werden.

Betrachten wir im folgenden, wie viele Änderungen im Mittel maximal hinzukommen.

1. $\frac{\binom{n}{0} \cdot (\frac{1}{4} \cdot 2 + \frac{1}{2} \cdot 1) \cdot 4}{2^{l+2}}$

2. $\frac{\binom{n}{1}\cdot\frac{1}{2}\cdot 1\cdot 4}{2^{l+2}}$
3. $\frac{\binom{n}{2}\cdot\frac{1}{4}\cdot 1\cdot 4}{2^{l+2}}$
4. 0

Fasst man alles zusammen, so kann die mittlere Kippzahl berechnet werden.

$$R_a^{+2} \leq R_a + \frac{\binom{n}{0}\cdot(\frac{1}{4}\cdot 2+\frac{1}{2}\cdot 1)\cdot 4+\binom{n}{1}\cdot\frac{1}{2}\cdot 1\cdot 4+\binom{n}{2}\cdot\frac{1}{4}\cdot 1\cdot 4}{2^{l+2}}$$
$$= R_a + \frac{\binom{n}{2}+2n+4}{2^{l+2}}$$

Es ist dabei sinnvoll die Ungleichung $2^l \geq \binom{n}{2}+\binom{n}{1}+\binom{n}{0}$ einzuhalten. Andernfalls ist die Approximation sehr pessimistisch, da in den ersten 3 Fällen mehr Änderungen gezählt werden, als eigentlich möglich.

Bei den hier verwendeten Blocklängen lohnt sich aufgrund der zu geringen mittleren Kippzahl nur das +1, sowie das +2 Schema, kein +3 Schema.

Eine Übersicht über die mögliche Effizienzverbesserung von Trelliskodes nach diesem Schema ist in Abbildung 3.13 dargestellt. Hier ist gut zu sehen, dass die Effizienz bei kurzen Kodes mehr gesteigert werden kann als bei längeren, da sich ein Bit Änderung hier stärker auswirkt.

3.7 Diskussion weiterer nicht praktikabler Möglichkeiten

In diesem Abschnitt sollen weitere Ideen vorgestellt werden, wie spezielle Eigenschaften von Trelliskodes für die Steganographie genutzt werden könnten. Zu jeder Idee soll begründet werden, warum sie so nicht durchführbar ist.

Variabler Startzustand mit Terminierung Nachdem in Abschnitt 3.4 vorgestellt wurde, wie eine Kodierung stattfinden kann ohne sich auf einen Startzustand festzulegen, besteht die Möglichkeit, diese Kodierungsform so zu modifizieren, dass auf gleiche Weise kodiert wird, aber noch Terminierung hinzugefügt wird um die dadurch hinzugefügte Redundanz für die Fehlerkorrektur zu benutzen und so die Einbettungseffizienz zu erhöhen.

Eine Terminierung in einen bestimmten Zustand fügt dem Wort immer so viel Bit Redundanz hinzu, wie das Gedächtnis groß ist. Da wir versuchen kurze Kodewortlängen zu erreichen und versuchen die in Abschnitt 3.2 aufgestellte untere Schranke mit Gleichheit zu erfüllen, verringert sich die Einbettungsrate durch eine Terminierung im Nullzustand erheblich.

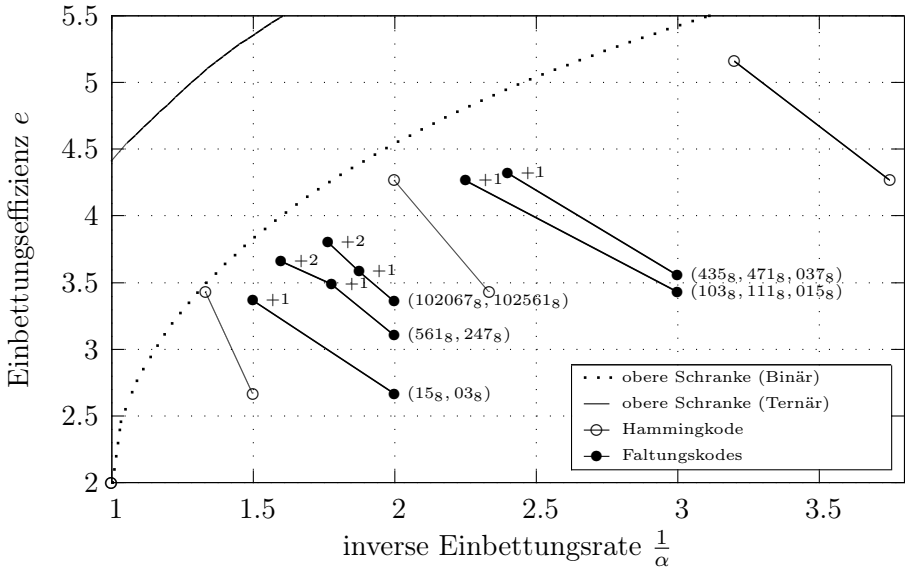


Abbildung 3.13: Übersicht über die mögliche Effizienzverbesserung von Trelliskodes mit dem +1/+2 Schema

Beispiel 15 Angenommen wir benutzen zwei Kodierer mit Gedächtnis 8. Um eine Terminierung in den Nullzustand durchzuführen benötigen wir also 8 zusätzliche Bits. Der erste soll eine Koderate von $\frac{1}{2}$, der zweite eine von $\frac{1}{3}$ haben. Aufgrund der Ungleichung 3.1, besitzt unser Quellkodewort für den ersten Kodierer eine Länge von genau 8 Bit. Für den zweiten darf die Länge zwischen 4 und 8 gewählt werden, doch auch hier ist es aus Effizienzgründen sinnvoll die untere Schranke mit Gleichheit zu erfüllen.

Die Einbettungsrate für den Kodierer mit der Koderate $\frac{1}{2}$ sinkt nun aufgrund der zusätzlichen 8 Bit auf $\frac{8}{2 \cdot 8+8} = \frac{1}{3}$. Für den Kodierer der Koderate $\frac{1}{3}$ sinkt die Koderate sogar auf $\frac{4}{3 \cdot 4+8} = \frac{1}{5}$. □

Wie im vorangegangenen Beispiel zu sehen ist, wird aufgrund der Tatsache, dass kleine Quellkodewortlängen verwendet wurden durch die Terminierung die Einbettungsrate stark verringert. Um bei so geringen Einbettungsraten mit vergleichbaren Kodes stand zu halten müsste die Terminierung allerdings eine sehr stark positive Auswirkung auf die Einbettungseffizienz haben. Um diese zu steigern, würde nun aber wieder eine Methode nach dem Prinzip verwendet werden, wie sie in Abschnitt 3.1 beschrieben ist. Diese kann jedoch aus schon in selbem Abschnitt genannten Gründen nicht die gewünschte Steigerung der Einbettungseffizienz bringen.

Tail-Biting Im Gegensatz zur Terminierung fügt Tail-Biting keine zusätzliche Redundanz hinzu.[16] Es ist daher ein naheliegender Gedanke, Tail-Biting statt Terminierung zu verwenden, um durch Fehlerkorrektur die Einbettungseffizienz zu erhöhen. Da man sich mit Tail-Biting allerdings auf den Startzustand festlegen muss, besteht keine Möglichkeit mehr über den Startzustand zu variieren. Eine Kodierung könnte daher nicht mehr die Vorteile des in Abschnitt 3.2 vorgestellten Verfahrens nutzen und hätte zusätzlich noch mit den Nachteilen zu kämpfen, wie sie anfangs in Abschnitt 3.1 beschrieben sind.

Natürlich wäre es möglich sich nicht auf einen Startzustand festzulegen, sondern nur auf einen Teil der Startzustände und aus diesem Teil die Trelliswörter auszuwählen. Allerdings bringt auch das aus steganographischer Sicht mehr Nachteile durch die Rückverfolgbarkeit der Fehlerkorrektur wie im selben Abschnitt beschrieben.

Soft-input Es existieren zwar Arbeiten, die einen Soft-input auch bei verschiedenen klassischen Kodes zulassen, bei der Dekodierung mit Trelliskodes ist allerdings eine Verarbeitung quantisierter Werte mit dem Viterbi Algorithmus ohne weiteres möglich. Ein Algorithmus, der den Soft-input zur steganographischen Kodierung nutzt ist in Abbildung 3.14 dargestellt und könnte so aussehen wie in Abschnitt 3.1 beschrieben mit dem Unterschied, dass nicht einzelne Bits als Wort zusammengefasst werden,

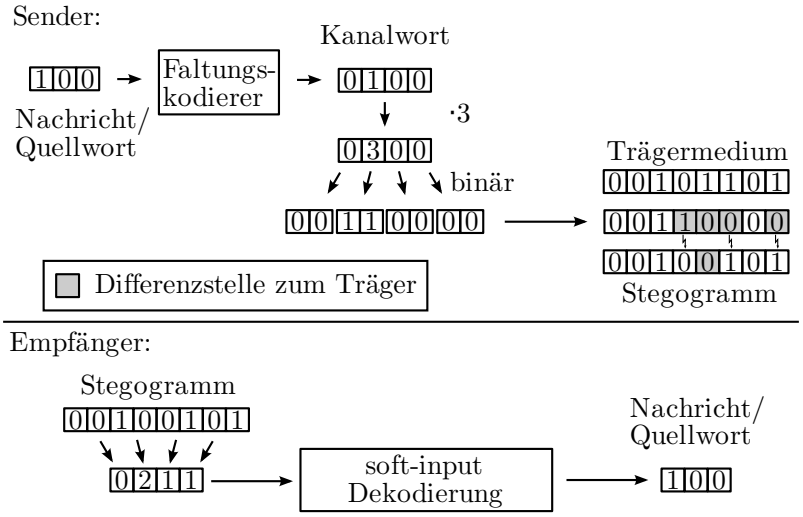


Abbildung 3.14: Kodierung mit Soft-input

sondern immer mehrere niederwertigste Bits eine Quantisierungsstufe eines Wortes darstellen. Werden dann gezielt Störungen im Kanalkodewort vorgenommen so kann der Sender entscheiden, welches Bit der jeweiligen Quantisierungsstufe gekippt wird.

[16] vgl. Abschnitt 2.2.1

Dabei entsteht eine stärkere Änderung, wenn der Sender das höherwertigere Bit der Quantisierungsstufe kippt, womit dem Sender mehr Spielraum gegeben wird.

Leider verringert sich die Einbettungsrate mit diesem Schema enorm. Werden 2 Bit zu einer Quantisierungsstufe zusammengefasst, so halbiert sich die Einbettungsrate, fasst man 3 Bits zusammen, so drittelt sich die Einbettungsrate usw..

Wie schon bei der Terminierung und in Abschnitt 3.1 diskutiert wurde, kann die Effizienz nicht wesentlich durch Fehlerkorrektur verbessert werden. Im Gegenzug bekommt man eine weitere Angriffsmöglichkeit, durch Analyse über die Anzahl der Wörter, die gerade noch zu einem Wort dekodiert werden (und keinen weiteren Fehler mehr zulassen).

4 Zusammenfassung und Ausblick

4.1 Zusammenfassung

Es wurde gezeigt, warum eine steganographische Kodierung nicht über eine Fehlerkorrektur statt finden kann. Anschließend wurde ein Verfahren aufgezeigt, wie mit Trelliskodes, speziell mit Faltungskodes, Nachrichten steganographisch kodiert werden können um sie anschließend in ein Trägermedium einzubetten. Zu diesem Verfahren wurden theoretische Betrachtungen durchgeführt, welche Grenzen eingehalten werden müssen und wie eine effiziente Implementierung statt finden kann. Anschließend wurde das Verfahren implementiert und durch Testreihen bestätigt.[1]

Zu diesem Verfahren wurden Verbesserungen aufgezeigt, wie es mittels Einbeziehung von Informationen aus dem Startzustand, rekursiven Faltungskodes und dem $+1/+2$ Schema weiter verbessert werden kann. Auch diese Verfahren wurden implementiert um deren Korrektheit zu bestätigen.

Weitere mögliche Verfahren wurden vorgestellt und ihre Unpraktibilität diskutiert. Eine Grafik, die einen Überblick über die vorgestellten Verfahren zeigt ist in Abbildung 4.1 dargestellt. Hier ist exemplarisch an 4 Kodes zu sehen, in welche Richtungen Verbesserungen möglich sind. Die Auswahl über den Kode entscheidet dann über die Berechnungskomplexität so wie es in Abschnitt 3.2 diskutiert wurde.

4.2 Ausblick

Das Thema Einbettung mit Trelliskodes ist mit dieser Arbeit keineswegs erschöpfend erfasst. Im wesentlichen haben zwei Faktoren den Umfang dieser Arbeit beschränkt.

Der erste maßgebende Faktor war die zeitliche Beschränkung. Die Idee, mehr als zwei abgehende Pfade von einem Zustand zuzulassen konnte nicht näher untersucht werden. Des Weiteren wäre noch eine genauere Betrachtung möglich, welche Bedingungen eine willkürliche Kantenbeschriftung haben muss, um eine eindeutige Dekodierung zuzulassen.[2] Mit diesen Bedingungen sollte experimentiert werden, ob die Einbettungseffizienz gesteigert werden kann. Auch wäre es denkbar die Kantenbeschriftung mehrerer Faltungskodierer in einem Trellisdiagramm darzustellen. Noch eine ungeklärte Frage ist, ob klassische Blockkodes, welche als Trellisdiagramm dargestellt werden einen Vorteil für die Einbettungseffizienz bringen.

[1] Der Quellcode ist auf einer CD dieser Arbeit beigelegt

[2] vgl. Abschnitt 3.5

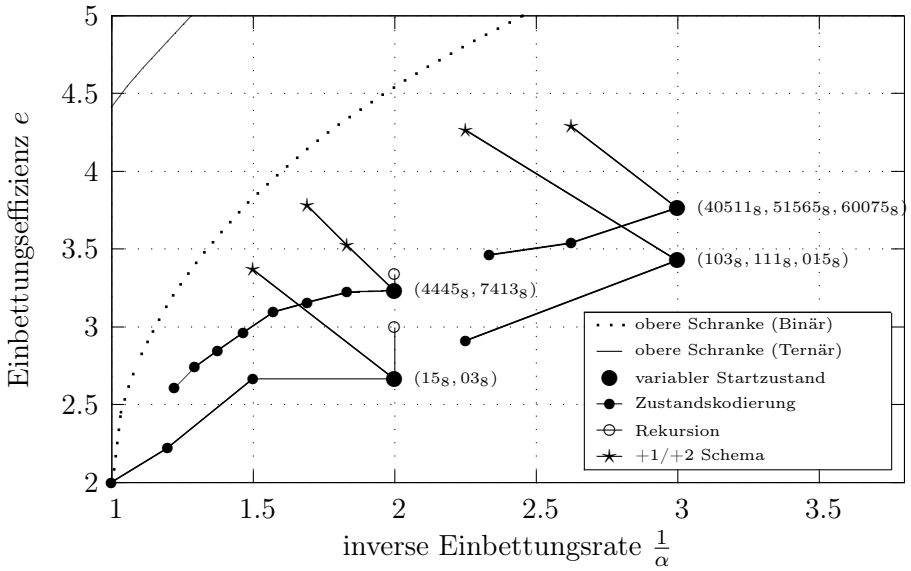


Abbildung 4.1: Überblick über die vorgeschlagenen Verfahren

Der zweite maßgebende Faktor waren die Ideen für weitere mit Trelliskodes durchführbare Verfahren. Gerade durch die Soft-decision Dekodierung unterscheiden sich Trelliskodes von klassischen Blockkodes und gerade durch diesen Punkt haben sich Trelliskodes in der Kanalkodierung so gut etabliert. Leider wurde kein überzeugendes steganographisches Kodierungsschema gefunden, was diesen Vorteil ausnutzt.

Literaturverzeichnis

Bahl, L. et al., Optimal decoding of linear codes for minimizing symbol error rate. IEEE Transactions on Information Theory, 20 Mar. 1974:2, S. 284–287.

Bossert, Martin, Kanalcodierung (Teubner Informationstechnik). Teubner Verlag, 1998, ISBN 3519161435.

Crandall, Ron, Some Notes on steganography. Posted on Steganography Mailing List, Dezember 1998 ⟨URL: `http://os.inf.tu-dresden.de/~westfeld/crandall.pdf`⟩.

Elias, Peter, Coding for Noisy Channels. IRE Conv. Rec. 4 1955, S. 37–46.

Esen, E., Alatan, A.A. und Askar, M., Trellis coded quantization for data hiding. in: EUROCON 2003. Computer as a Tool. The IEEE Region 8. Band 2, September 2003, S. 384–388vol.2.

Fridrich, J., Goljan, M. und Soukal, D., Wet paper codes with improved embedding efficiency. Information Forensics and Security, IEEE Transactions on, 1 2006:1, S. 102–110 ⟨URL: `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1597139`⟩.

Fridrich, Jessica und Filler, Tomáš, Practical Methods for Minimizing Embedding Impact in Steganography. in: Security, Steganography, and Watermarking of Multimedia Contents IX part of Electronic Imaging, 28 January-1 February 2007, San Jose, USA | SPIE proceedings Volume 6505. Februar 2007.

Hamming, R. W., Error detecting and error correcting codes. The Bell System Technical Journal, 29 April 1950:2, S. 147–160, Reprinted in E. E. Swartzlander, *Computer Arithmetic*, Vol. 2, IEEEComputer Society Press Tutorial, Los Alamitos, CA, 1990.

Klimant, Herbert, Piotraschke, Rudi und Schönfeld, Dagmar, Informations- und Kodierungstheorie. Teubner, 2006, ISBN 3835100424.

Lin, Lin et al., An efficient algorithm for informed embedding of dirty-paper trellis codes for watermarking. in: ICIP (1). 2005 ⟨URL: `http://dblp.uni-trier.de/db/conf/icip/icip2005-1.html\#LinDCM05`⟩, S. 697–700.

Miller, M.L., Doërr, G.J. und Cox, I.J., Dirty-paper trellis codes for watermarking. in: Image Processing. 2002. Proceedings. 2002 International Conference on. Band 2, September 2002, S. II–129–II–132vol.2.

Miller, M.L., Doërr, G.J. und Cox, I.J., Applying informed coding and embedding to design a robust high-capacity watermark. Image Processing, IEEE Transactions on, 13 Juni 2004:6, S. 792–807.

Palazzo, R.P., Jr., A network flow approach to convolutional codes. Communications, IEEE Transactions on, 43 Februar -März -April 1995:234, S. 1429–1440.

Schönfeld, Dagmar und Winkler, Antje, Embedding with syndrome coding based on BCH codes. in: MM&Sec '06: Proceeding of the 8th workshop on Multimedia and security. New York, NY, USA: ACM Press, 2006, ISBN 1–59593–493–6, S. 214–223.

Viterbi, A. J., Error Bounds for Convolutional Codes and an Asymptotically Optimal Decoding Algorithm. IEEE Trans. Inform. Theory, 13 April 1967:2, S. 260–269.

Westfeld, Andreas, F5-A Steganographic Algorithm. in: IHW '01: Proceedings of the 4th International Workshop on Information Hiding. London, UK: Springer-Verlag, 2001, ISBN 3–540–42733–3, S. 289–302.

Westfeld, Andreas und Pfitzmann, Andreas, Attacks on Steganographic Systems. in: **Pfitzmann, Andreas (Hrsg.):** Information Hiding. Band 1768, Springer, 1999, ISBN 3–540–67182–X, S. 61–76.

Zhang, Weiming, Wang, Shuozhong und Zhang, Xinpeng, Improving Embedding Efficiency of Covering Codes. Communications, IEEE Transactions on, 11 August 2007:8, S. 680–682.

Danksagung

Am meisten möchte ich mich bei meiner Betreuerin Dagmar Schönfeld bedanken. Hätte sie nicht so beharrlich gebohrt und nachgefragt, so hätte ich schon sehr früh aufgegeben.

Weiterer Dank gebühren Kathleen Ebel und Immanuel Scholz für die unzähligen kleinen Korrekturen.

Bei meiner Frau Tina will ich mich bedanken, dass sie mir immer den Rücken frei hielt.

Auch Arthur und Oskar will ich danken, dass sie mir frei gaben und so oft allein Sandmann schauten.

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die von mir eingereichte Diplomarbeit vollkommen selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Dresden, den 14. Dezember 2007 Benjamin Kellermann