Technische Universität Dresden

Großer Beleg

Implementierung von Matrixkodierung auf Basis des steganographischen Algorithmus F5

Verfasser:	Marcel Läzer
Studienrichtung:	Informatik
Betreuer:	Dr.-Ing. Andreas Westfeld
Hochschullehrer:	Prof. Dr. rer. nat. Andreas Pfitzmann
	Institut für Systemarchitektur

Dresden, im Juli 2001

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einführung

In Zeiten der globalen Vernetzung, da mehr und mehr mitunter sehr persönliche Daten ihren Weg über etliche fremde Rechner zum Ziel suchen, ist es unerläßlich, die Kommunikation vor unberechtigtem Zugriff zu schützen. Wer möchte schon, daß seine Liebesbriefe, Kontoauszüge, Ergebnisse ärztlicher Untersuchung usw. von neugierigen Nachbarn, dem Arbeitgeber, Diktatoren, Cyberkriminellen oder wem auch immer ausgewertet werden.

Eine Lösung ist Kryptographie, die beliebige Daten zu scheinbarem Zeichenchaos verschlüsselt, das nur noch vom Inhaber des passenden Schlüssels verstanden werden kann. Leistungsfähige Kryptoprogramme[1] sind für die meisten Plattformen kostenfrei erhältlich. Leider versuchen Regierungen immer wieder, Kryptographie zu verbieten, oder sie verabschieden Gesetze, die die Beweispflicht umkehren und potentiell jeden, der auf elektronischem Wege mit seiner Umwelt in Kontakt tritt, zum Kriminellen stempeln. Unschuld muß ggf. durch Herausgabe besagten Schlüssels bewiesen werden.

Eine weitere, gerade unter obigen Bedingungen interessante Lösung ist Steganographie (griechisch für „verdeckt Schreiben“). Anstatt die Nachricht mittels „rasselnder Ketten“ zu sichern, wird sie hier still und leise in Wirtsdaten „versteckt“, deren Versand legal und plausibel ist. Somit bleibt nicht nur der Inhalt der Kommunikation verborgen, sondern auch die Tatsache, daß überhaupt kommuniziert wird. Einziger Nachteil ist das erhöhte Datenaufkommen durch den Wirt. Natürlich kann die Nachricht zuvor zusätzlich kryptographisch gesichert werden, womit maximale Sicherheit erreicht wird.

1.1 Grundlagen

Der als sicher geltende Algorithmus **F4** [4], auf den diese Arbeit aufsetzt, bettet die geheime Nachricht in JPEG[2]-Dateien ein, wie sie häufig verwendet werden, um Truecolour-Grafiken mit einstellbarer Bildqualität zu verschicken. Neben einem für Steganographie uninteressanten Kopf bestehen sie aus einem Rumpf, der entropiekodierte[3] Koeffizienten der Diskreten Kosinus-Transformation, im folgenden „DCTK“ genannt, enthält. Diese spiegeln den Frequenzraum des Bildes wieder, wobei dieses nicht als Ganzes transformiert wird, sondern seine Zerlegung in Unterblöcke aus je 8×8 Bildpunkten. Durch mehr oder weniger starke Rundung der Koeffizienten mittels $DCTK_{x,y} := f_{x,y}(qualitaet)\,\lfloor DCTK_{x,y}/f_{x,y}(qualitaet)\rfloor$, die über den Parameter *qualitqaet* nahezu frei einstellbar ist, sinkt zwar die Bildqualität, im Gegenzug steigt aber die Kompressibilität, da die relativen Häufigkeiten zugunsten der betragsmäßig

[1] beispielsweise PGP (“pretty good privacy” , `www.pgpi.org`)

[2] Grafikformat der Joint Photographic Experts Group, `www.jpeg.org`

[3] die Symbole einer Menge werden entsprechend ihrer Auftrittswahrscheinlichkeit mit mehr oder weniger Bits kodiert. Wirkt am besten bei Verteilung der Art 1:1:2:4:8:16...

kleinen Koeffizienten, insbesondere der 0, verschoben werden. Geringe Änderungen an einigen (außer der Nullfrequenz[4] potentiell allen) DCTK eines Unterblocks erzeugen keine spezifischen Artefakte und sind visuell[5] nicht nachweisbar.

Der **F4**-Algorithmus bettet ein Bit pro DCTK in dessen niederwertigstes Bit („LSB“) ein, indem jener betragsmäßig dekrementiert wird, falls es nicht schon den richtigen Wert enthält. Damit wird grob eine Verringerung des Qualitätsparameters modelliert. Zwar unterscheiden sich die Resultate, handelt es sich bei dem einen doch um eine additive, bei dem anderen aber um eine multiplikative Änderung, allerdings konnte Jens Zeidler in seiner Diplomarbeit [6] zeigen, daß die Unterschiede zu gering sind, als daß sich daraus statistische[6] Angriffe ableiten ließen.

Wichtig dabei ist, daß in Koeffizienten, die 0 sind, („DCTK_0“) nicht eingebettet werden kann und daß solche, die im Falle einer Änderung zu 0 werden dann auch keine Information mehr tragen können, was als Schwund bezeichnet wird. Letzteres betrifft die DCTK mit dem Wert $+1$ oder -1 („DCTK_1“). Mit “Bett” sei der Vektor der durch den Algorithmus **F4** beschreibbaren Bits der Hülle bezeichnet. Er wird einer vom Schlüssel abhängigen Permutation unterworfen. Damit werden einerseits die Einbettungsstellen über die gesamte Hülle verstreut (bei kurzen Nachrichten), andererseits wird der öffentliche Algorithmus parametrisiert.

1.2 Motivation

Es scheint also eine optimale Einbettungsfunktion gefunden, jedoch sind trotzdem noch Verbesserungen denkbar. Erstens ist es schade um ungenutzte Bits, denn nur selten wird eine Hülle bereitstehen, deren Bett die Botschaft genau aufnimmt. Zweitens ist, bei aller Sicherheit dieser Einbettungsfunktion, die Wahrscheinlichkeit, einen korrekten Verdacht bezüglich steganographischer Bearbeitung einer Datei zu äußern, umso größer, je mehr Änderungen vorgenommen wurden. Ein einfaches Beispiel soll zeigen, daß es möglich ist, überschüßige Bettkapazität zur Verringerung der Änderungsanzahl zu verwenden: möchte man in 8 Bits Bett 8 Bits einbetten, sind im Extremfalle auch 8 Bits zu ändern (Die Wahrscheinlichkeit hierfür beträgt $^1/_{256}$, für 7 Bits $^{28}/_{256}$ ...). Spendiert man nun ein zusätzliches Bit, das die Bedeutung aller ursprünglichen Bits negieren kann, sind nur noch maximal $\frac{8}{2}$ Bits zu ändern. Abbildung 1 zeigt ein Beispiel.

Die Möglichkeiten, die eine geschickte Kodierung bietet, sind in Abbildung 2 dargestellt. Ron Crandall hat hier die Bezeichnung „$(d_{\max}, n, k)$-Matrixkodierung“ eingeführt: durch

[4] Bei dieser heben sich positive und negative Elongationen nicht gegenseitig auf.
[5] grafische Darstellung aller potentiellen Stego-Bits.
[6] hauptsächlich Bewertung des Histogrammes aller DCTK

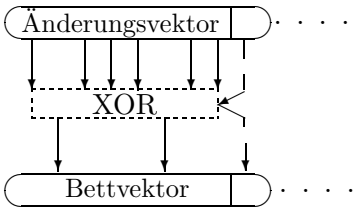


Abbildung 1: Verringerrung der Anzahl nötiger Änderungen

höchstens $d_{\max}$ Änderungen lassen sich in n Bits bei beliebiger Ausgangssituation k Bits kodieren [1]. Die beiden Diagramme zeigen die Anzahl $D_{\max}$ nötiger Änderungen in Abhängigkeit von der Bettgröße N und der Nachrichtengröße K für a) naive Kodierung, b) optimale Kodierung.

Ziel ist also, sich möglichst weit an dieses Optimum anzunähern. Dazu wird eine Systematik der Kodes entwickelt und eine Vorschrift zur Auswahl der besten Parameter für eine beliebig[7] gegebene Kombination aus Bett und Botschaft vorgestellt werden. Weiter wird versucht, schnelle[8] Algorithmen für die Umsetzung der Kodes zu finden.

2 Matrixkodierung

Als erstes gilt es, die Abhängigkeiten zwischen der Größe N des Bettes , der Größe K der Nachricht und der zulässigen Änderungsanzahl $D_{\max}$ festzustellen sowie die wichtigsten Voraussetzungen für die Existenz eines passenden Matrixkodes anzugeben. Sei B die Menge aller durch das Bett darstellbaren Vektoren (also das Kreuzprodukt $\{0,1\}^N$), M sei Menge aller möglichen Botschaften $\{0,1\}^K$. Die Menge E enthalte alle gültigen Änderungsvektoren - dies sind jene Differenzvektoren zwischen Elementen aus B, deren Hamminggewicht[9] (das ist die Anzahl gesetzter Bits in einem Wort/Vektor) $D_{\max}$ nicht übersteigt. Ein gesetztes Bit in einem solchen Vektor bedeutet Korrektur eines DCTK und bringt die Änderung potentiell mehrerer Hüllenbits mit sich, was für unsere Betrachtungen jedoch nicht von Bedeutung ist. Auch die Verteilung spielt keine Rolle - alleiniges Maß ist das Hamminggewicht. (Prinzipiell werden die in dieser Arbeit gewonnenen Erkenntnisse allen Stegosystemen zugute kommen, nur muß die Menge E eventuell anderen Kriterien wie z. B. minimalem Rauschen oder maximaler Bildkorrelation angepaßt werden.)

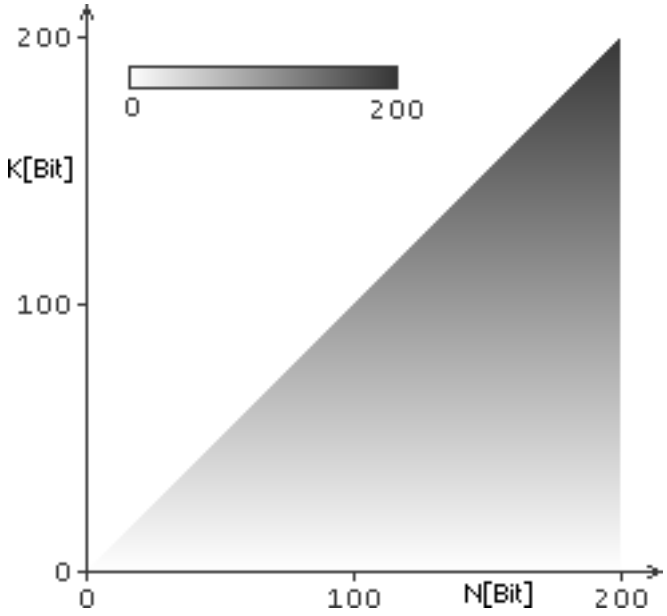
Um einen gültigen Kode zu erhalten, ist es notwendig, von jedem gegebenen $\overrightarrow{b}$ aus durch Addition[10] eines $\overrightarrow{e}$ in jede beliebige Restklasse wechseln zu können. Daraus leitet sich unmittelbar

[7] natürlich $K \leq N$

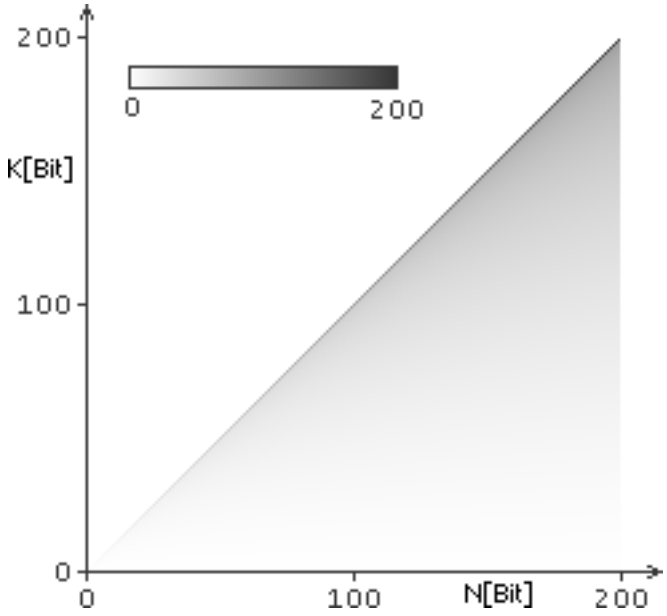
[8] das tollste Stegosystem nützt nichts, wenn die Akzeptanz fehlt. Bisher bekannte Algorithmen haben exponentielle Komplexität.

[9] Eigentlich wäre Hammingmasse korrekt.

[10] die bitweise Modulo-2-Addition, d.h. es gibt keinen Übertrag zwischen den Binärstellen



a) bei naiver Kodierung



b) informationstheoretisch minimal

Abbildung 2: Anzahl D nötiger Änderungen

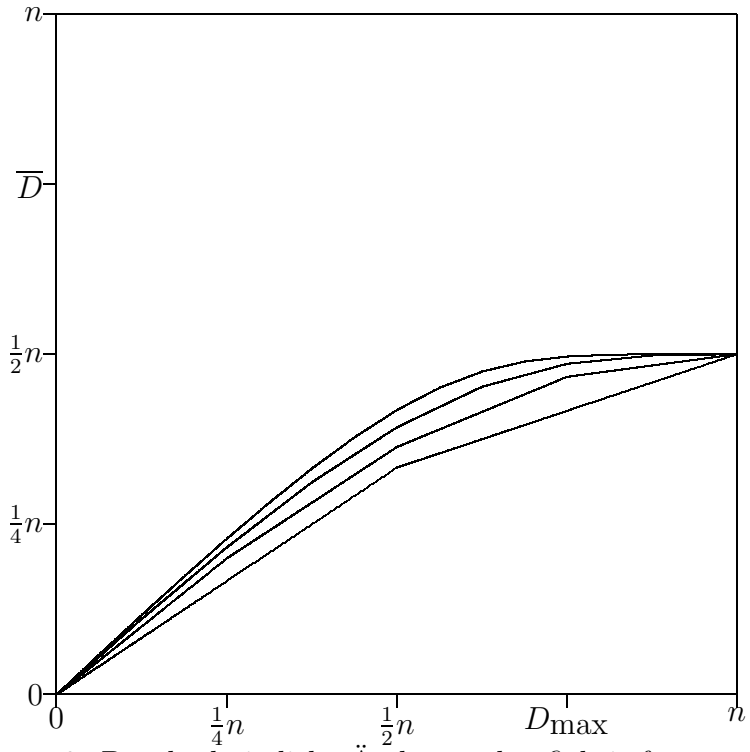


Abbildung 3: Durchschnittliche Änderungshäufigkeit für $n = 2, 4, 8, 16$

die Forderung ab, daß es mindestens so viele Änderungsvektoren wie mögliche Nachrichten geben muß. Da die Anzahl der Vektoren der Länge v mit genau w gesetzten Bits $\begin{pmatrix} v \\ w \end{pmatrix}$ ist, muß also folgende Ungleichung erfüllt sein:

$$\sum_{i=0}^{D_{max}} \begin{pmatrix} N \\ i \end{pmatrix} \geq 2^K \tag{1}$$

Unter Umständen ist eine rekursive Berechnung der Summe sinnvoll:

$$\sum_{i=0}^{D_{max}} \begin{pmatrix} N \\ i \end{pmatrix} = S(N, D) = \begin{cases} 1 & N \cdot D = 0 \\ S(N-1, D) + S(N-1, D-1) & \text{sonst} \end{cases}$$

Tritt in Gleichung 2 Identität ein, so spricht man von einem perfekten Kode. Die Klassenführer sind dann jeweils kleinstes Element, und $E = \bigcup_{i=0}^{2^K-1} h^{-1}(i)$

Für Einbettungsfunktionen mit QOS[11]-Anforderungen an die Änderungsanzahl ist $D_{\max}$ das einzig interessante Maß. Bei **F4** ist dies aber nicht der Fall, und so lohnt es sich, eine Betrachtung über die durchschnittliche Anzahl $\overline{D}$ anzustellen.

Abbildung 3 verdeutlicht den Zusammenhang. (Kommen keine perfekten Kodes zum Einsatz, kann $\overline{D}/D_{\max}$ noch geringer ausfallen, vorausgesetzt, der Kode nutzt aus dem übergroßen Angebot an Änderungsvektoren eben jene mit dem geringsten Gewicht.)

[11] Quality Of Service - Zusicherung einer Eigenschaft

Dies soll vor allem dazu dienen, die tatsächlichen Auswirkungen von $D_{\max}$ richtig einzuschätzen, denn es ist ebendieses $D_{\max}$, anhand dessen auch im folgenden alle Kodes bewertet werden.

So sei an dieser Stelle noch die Kodequalität Q definiert. Sie basiert auf dem von einem bestimmten (d, n, k)-Kode für ein bestimmtes N, K geforderten $D_{\max}$. Es erfolgt dabei eine lineare Bewertung zwischen dem (K, K, K)-Kode mit 0% und dem informationstheoretisch optimalen (D_0, N, K)-Kode mit 100%:

$$Q(d,n,k,N,K) = \begin{cases} 1 & K = D_0 \\ -0.1 & \text{es existiert kein solcher Kode } [12] \\ \frac{K - z \cdot d}{K - D_0} & \text{sonst} \end{cases}$$

Weiter sei $Q_N(d,n,k) = \sum_{i=1}^{N} \frac{Q(d,n,k,N,i)}{N}$, die durchschnittliche Qualität eines Kodes für eine gegebene Bettgröße.

2.1 Reine Kodes

Im allgemeinen wird vermieden, Nachrichten als Ganzes zu kodieren[13]. Dies hat zum einen komplexitätstheoretische Gründe („divide et impera“), zum anderen erlaubt es den wiederholten Einsatz einiger handoptimierter Kodes. Hier kommt noch hinzu, daß sich durch Schwund die Ergebnisse komplizierter Rechnung am Ende als nicht umsetzbar erweisen können.

Deshalb sind Nachricht und Bett auf mehrere Kodeworte aufzuteilen, deren Anzahl mit z bezeichnet sei. (D, N, K) gehen dann in (d, n, k) über, wobei folgende Bedingungen erfüllt sein müssen:

$$\begin{aligned} i: &\quad zk \geq K \\ ii: &\quad zn \leq N \end{aligned}$$

In aller Regel wird $zd > D$ sein; das ist der Preis für die Vereinfachung.

Ein sehr einfacher Kode ist der $(1, 3, 2)$-Kode [1]. Von diesem ausgehend lassen sich zwei unendliche Mengen trivialer Kodes ausmachen: zum einen die in der Einführung erwähnten $(d, 2d+1, 2d)$-Kodes und die $(1, 2^k - 1, k)$-Kodes[14]. Erstere halbieren $D_{\max}$ zwar gegenüber naiver Kodierung, die durchschnittliche Anzahl an Änderungen $\overline{D}$ wird dadurch aber kaum verringert (s. Abbildung 3). Letztere sind dagegen wesentlich interessanter. Sie wurden schon von Andreas Westfeld für den verbesserten Algorithmus F5 sehr effizient implementiert [5]. Abbildung 4 zeigt die Qualität dieser Kodes (bei idealer Parameterwahl) bis $N = 200$. Für

[12] willkürlich festgelegte Strafbewertung.
[13] auf Faltungskodes trifft das nicht zu ...
[14] Sie entsprechen den in der Kodierungstheorie bekannten Paritäts- und Hammingkodes [2]

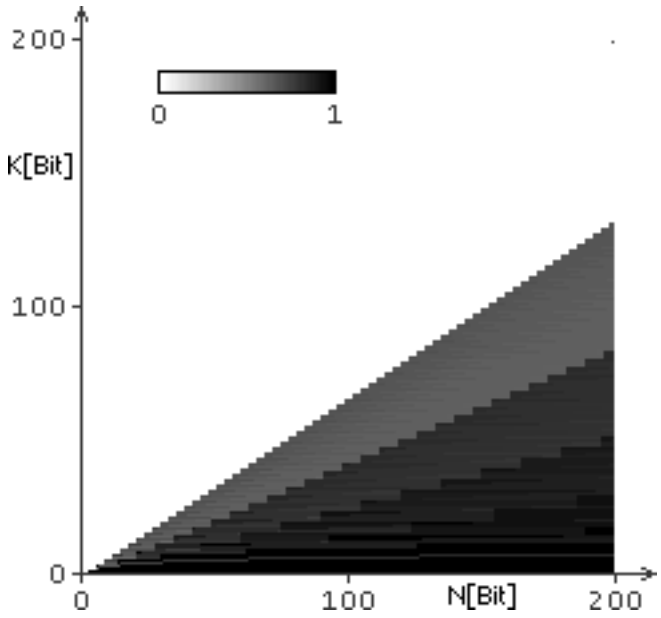


Abbildung 4: Qualität Q der $(1, n, k)$-Kodes

geringe $\frac{K}{N}$ sind sie nahezu optimal, und $Q_{N>40}(1, 2^k - 1, k) \approx 50\%$.

Durch das Zulassen größerer $D_{\max}$ steigt zwar die Änderungsdichte, die verbesserte Einbettungsrate sollte dies aber mehr als aufwiegen. Diagramm 5 zeigt einige Kodes zur Verdeutlichung (logarithmischer Maßstab!). Nun, dies sind auf den ersten Blick beachtliche Ergebnisse, doch man sollte nur Kodes innerhalb einer $d_{\max}$-Klasse vergleichen. Dazu sind in Diagramm 6 noch die entsprechenden $(d_{\max} \cdot (1, \frac{n}{d_{\max}}, k'))$- Kodes eingetragen. Es zeigt sich, daß die Verbesserungen $k - k'$ für ein beliebiges $d_{\max}$ jeweils ungefähr konstant sind, sich damit die Informationsraten im Unendlichen annähern. Außerdem wirkt sich die gröbere Quantelung der Kodeworte, die ein größeres $d_{\max}$ mit sich bringt, negativ auf die Anpaßbarkeit an gegebene N, K aus. Somit ist für kleine N (einige 100 bis 1000bit) ein Steigerung von $d_{\max}$ nicht ratsam, für niedrige Einbettungsraten lohnt sich der Aufwand nicht.

2.2 Untersuchung von Mischkodes

Wenn man noch einmal Abbildung 4 betrachtet, fällt die grobstufige Struktur auf; bei größeren $d_{\max}$ verstärkt sich diese Eigenschaft noch. Dies liegt an den Sprüngen von einem zum nächsten Kode, wobei sich die Kodeparameter deutlich ändern. Durch das Mischen von Kodes sollten sich die Kanten weitgehend glätten lassen. Welche Kodes sind nun zu mischen? Diagramm 7 zeigt die Abhängigkeit der Kodekapazität vom Verhältnis $n_0 : n_1$. Es ist also eine möglichst symmetrische Aufteilung anzustreben, die natürlich am ehesten durch jene beiden Kodes erreicht wird, die an die Sprungstelle angrenzen.

Die Hashmatrix H eines solchen $(d_{\max 0} + d_{\max 1}, n_0 + n_1, k_0 + k_1)$- Kodes läßt sich leicht aus

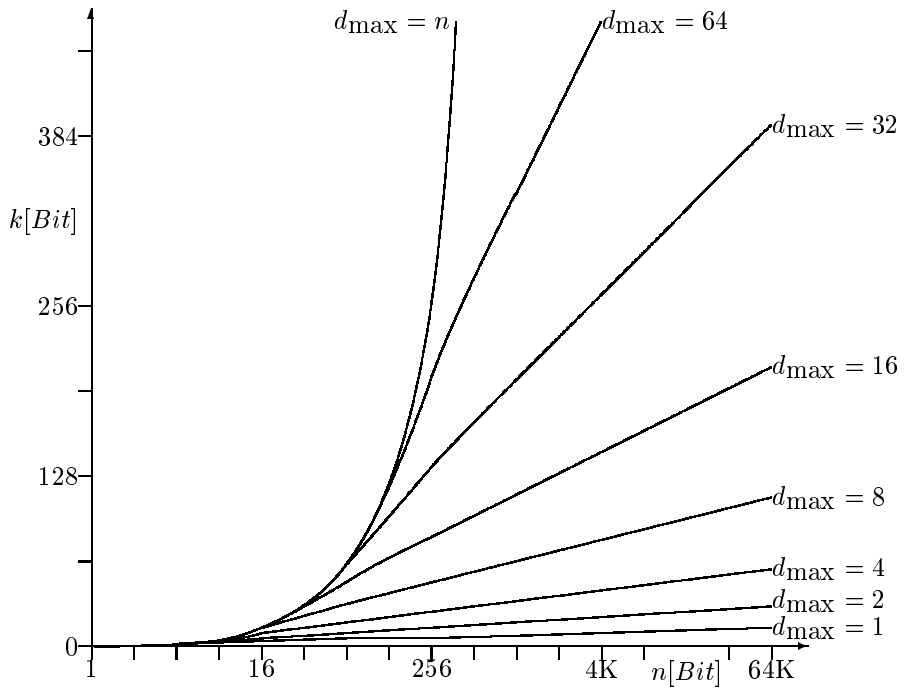


Abbildung 5: Kodewortkapazität k für einige $d_{\max}$-Klassen

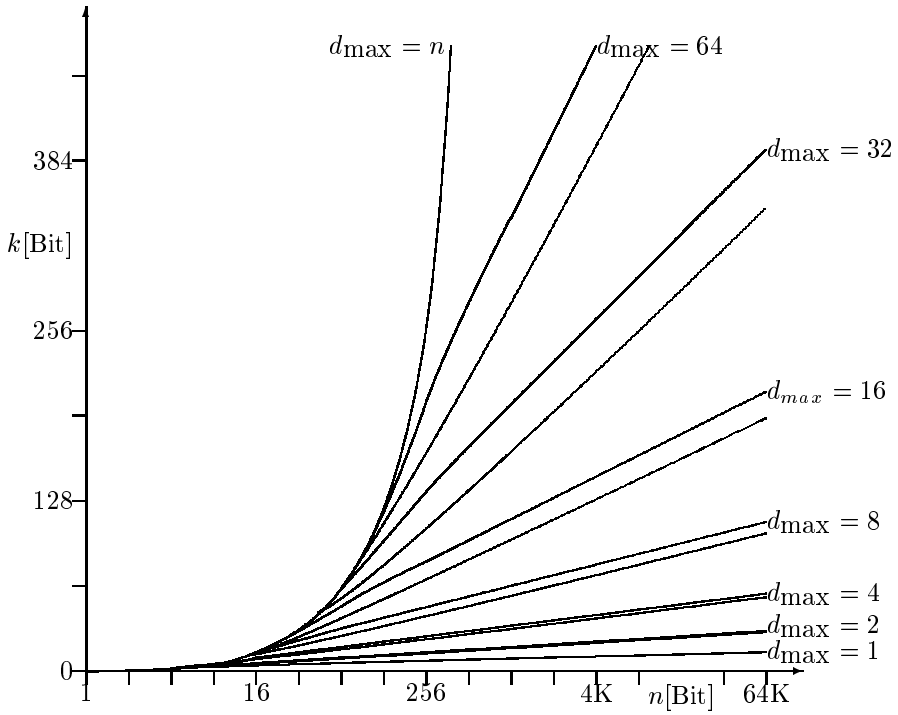


Abbildung 6: Kodewortkapazität k, Vergleich mit $(1, n', k')$-Kodes

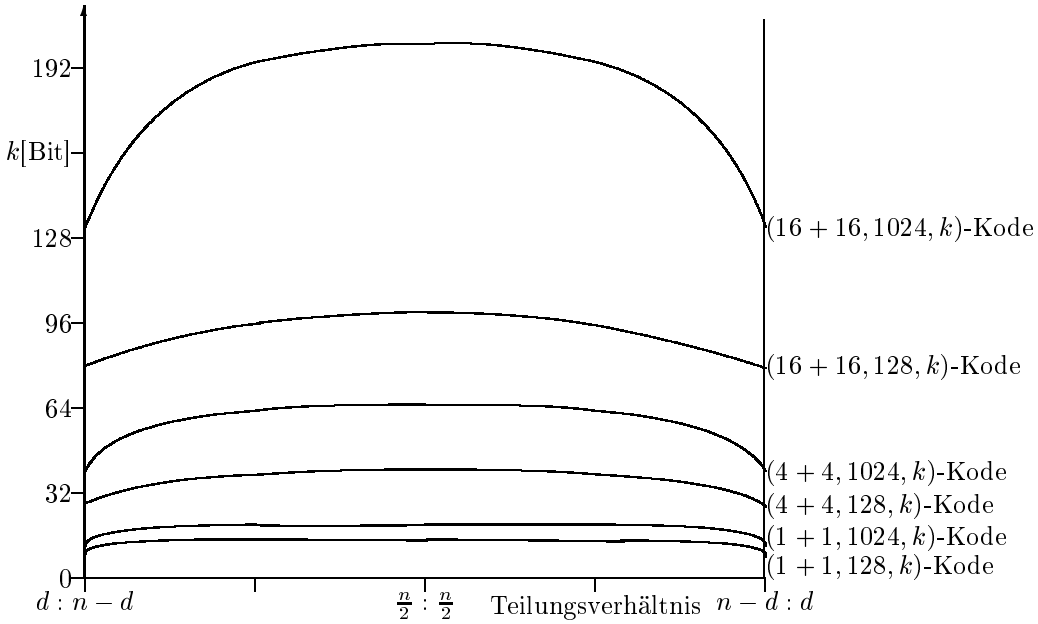


Abbildung 7: Kodekapazität k von Mischkodes

den Matrizen H_0, H_1 der zu mischenden Kodes erzeugen[15]:

$$H = \begin{pmatrix} H_0 & 0 \\ 0 & H_1 \end{pmatrix}$$

Die Komplexität der Kodierung ist dabei die Summe der Einzelkomplexitäten. Durch wiederholtes Mischen lassen sich Makrokodes erzeugen, mit denen die Größen N, K gut approximiert werden.

2.2.1 Mischkodes aus einer $d_{\max}$-Klasse

Es empfiehlt sich, Makrokodes aus nur zwei Basen aufzubauen und die Hashmatrix folgendermaßen zu ordnen:

$$H = \begin{pmatrix} H_0 & & & & \\ & \ddots & & 0 & \\ & & H_0 & & \\ & & & H_1 & \\ & 0 & & & \ddots \\ & & & & & H_1 \end{pmatrix} \begin{matrix} \left.\vphantom{\begin{matrix}a\\a\\a\end{matrix}}\right\} z_0 \cdot H_0 \\ \\ \left.\vphantom{\begin{matrix}a\\a\\a\end{matrix}}\right\} z_1 \cdot H_1 \end{matrix}$$

So lassen sich sukzessive die $z_0 + z_1$ (Basis-)Kodeworte unabhängig voneinander berechnen.

Beispielhaft sei ein solcher Mischkode aus $(1, n, k)$-Kodes bewertet (Abbildung 8).

[15] Eine der 0-Matrizen kann bei Bedarf beliebig ersetzt werden.

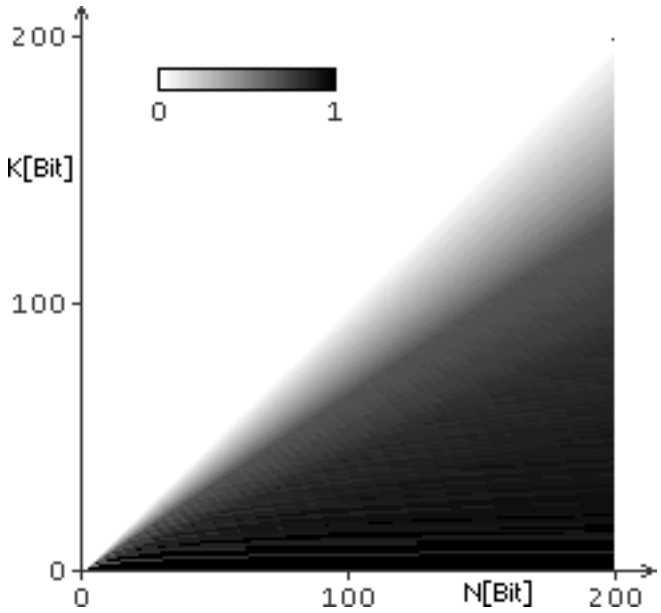


Abbildung 8: Qualität Q der $(1, n, k)$-Mischkodes

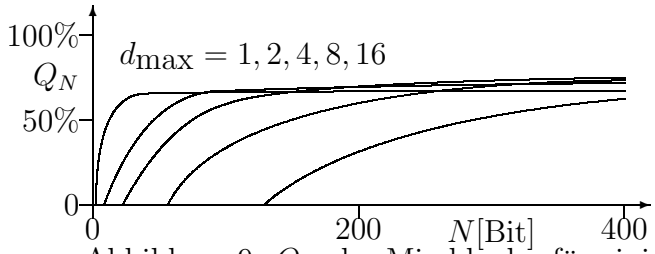


Abbildung 9: Q_N der Mischkodes für einige $d_{\max}$

Dieser Kode erreicht eine Qualität von 64 %. Die Optimierung von Kodes mit $d_{\max} > 1$ birgt noch ein größeres Potential, allerdings kann dieses erst bei großer Kodewortzahl erschlossen werden, wie aus Diagramm 9 ersichtlich ist. Für den $(1, n, k)$-Kode ist diese Verbesserung in jedem Falle lohnenswert, da der Aufwand sowohl für die Parameterbestimmung wie auch die Generierung der beiden Basiskodes vernachlässigbar ist.

2.2.2 Klassenübergreifende Mischkodes

Neben n, k kann natürlich auch $d_{\max}$ variiert werden. Allerdings verkompliziert dieser zusätzliche Freiheitsgrad die Bestimmung optimaler Parameter (es gibt wesentlich mehr als zwei Kodes, die dann zur Auswahl stehen). Die erreichbaren Verbesserungen basieren im wesentlichem darauf, daß N, K noch besser angenähert werden können, wobei der größte Fortschritt hier schon durch die Mischkodes einer Klasse geleistet wurde. Ist eine $d_{\max}$-Klasse beherrschbar, sollte man nicht auf leistungsschwächere Klassen zurückgreifen. Eine Ausnahme bilden die leicht berechenbaren $(d, 2d + 1, 2d)$-Kodes, deren $d_{\max}$-Klasse allgemein möglicherweise nicht beherrschbar ist. Sie sind bei hoher Einbettungsrate als Ergänzung zum $(1, 3, 2)$- und $(1, 1, 1)$-Kode nützlich.

2.3 Beziehung zu fehlerkorrigierenden Kodes

Die Ähnlichkeit zu (n, k, d)-fehlerkorrigierenden Kodes liegt nicht nur in der Notation.[16] Auch dort geht es darum, Kodeworte zu korrigieren und Nachrichten zu extrahieren. Während bei der Matrixkodierung aber die k Nachrichtenbits nur implizit durch die n (redundanzfreien) Kodebits übertragen werden, wobei alle 2^n Worte gültig sind, werden dort die k Nachrichtenbits innerhalb der n Kodebits übertragen und die übrigen $n - k$ Bits sind Redundanz, die eine paarweise Mindestdistanz d zwischen den (nur 2^kverschiedenen) Kodeworten garantieren. Alle zugelassenen Änderungen bewegen sich innerhalb der Restklasse (sog. „Korrekturkugel mit Radius $\left\lfloor \frac{d-1}{2} \right\rfloor$“) des gegebenen Kodewortes, während sie beim Matrixkode orthogonal dazu immer aus dieser Klasse heraus in die verschieden anderen Restklassen wechseln. Fehlerkorrigierende Kodes besitzen eine Funktion zum Testen der Redundanz, oft als „Kontrollmatrix“ dargestellt, die der Hashfunktion sehr ähnlich ist. Insbesondere die Theorien über ihre Generierung, wie sie beispielsweise in [3] beschrieben sind, können deshalb auch hier Anwendung finden.

3 Bestimmung der Kodeparameter

Das beste Ergebnis erreicht man natürlich mit einem (D_0, N, K)-Kode, wozu anhand Gleichung (1) für N, K ein minimales $D_{\max}$ zu bestimmen ist. (Iterativ ist das sehr einfach möglich mit ca. $ND_{\max}$ Additionen) Da dann aber die in Abschnitt 2.1 angeführten Probleme geballt auftreten werden, ist es besser, den Anwender zwischen schnellerer Abarbeitung oder besserer Kodierung wählen zu lassen.

Während der Rechenaufwand zum Auslesen (das auch Teil des Kodierens ist) als fest vorgegeben angenommen werden kann, denn es müssen immer ca. N Bits gelesen und K Bits geschrieben werden, hängt er beim Korrigieren maßgeblich von $d_{\max}$ ab. Dieser Parameter bestimmt also sowohl die erreichbare Kodequalität als auch den Aufwand und bietet sich damit als Stellgröße an.

Hat sich der Anwender für ein $d_{\max}$ entschieden, können daraus und den festen Größen N, K die freien Parameter n, k des Kodes bestimmt werden. Dies geschieht iterativ über k, bis jener Kode mit dem größtmöglichen k gefunden ist. Da Mischkodes zum Einsatz kommen sollen, ist noch das Mischverhältnis $z_0 : z_1$ zu bestimmen. Auch dies geschieht iterativ, indem soviele Kodeworte wie möglich durch den nächsten (den „ersten unmöglichen“) Kode ersetzt werden. Der Aufwand für beide Aufgaben ist vernachlässigbar.

[16] trotz gleicher Symbole hat das Tripel eine andere Bedeutung

4 Ermittlung des Änderungsvektors

Nun, da die bestmöglichen Parameter $d_{\max}, n, k$ gefunden sind, ist dazu ein passender Kode zu bestimmen. Ein solcher Kode muß zwei Dinge leisten: aus einem Kodewort die Nachricht extrahieren („hashen“) als auch ein Kodewort entsprechend einer gewünschten Nachricht zu ändern („korrigieren“).

Einige Definitionen: Die Abbildung $h : B \to M$ zerlegt die Menge B in 2^K Restklassen (möglichst) gleicher Mächtigkeit. Diese Restklassen entsprechen den möglichen Botschaften und seien mit $B_i , i \in N[0, 2^K)$ bezeichnet. Sie besitzen ein ausgezeichnetes Element, den Klassenführer, der minimales Element bezüglich des Hamminggewichtes ist. Die Abbildung h beschreibt den Kode vollständig. Die inverse Abbildung $h^{-1} : M \to B$ liefert zu jeder Restklasse den Klassenführer. Sie hat eher beschreibenden Charakter, da sie sich in der Realität nur selten explizit angeben läßt.

4.1 Grundlagen der Korrektur

Während das Hashen keine besondere Herausforderung darstellt (Modulo-Division, Matrizenmultiplikation, Verknüpfungsbäume eignen sich hier), ist es ein echtes Problem, schnell zu korrigieren. Deshalb wird die Kodierung im folgenden hauptsächlich von dieser Seite betrachtet.

Neben der fundamentalen Frage der zu verwendenden Datentypen (wenn möglich, wird man sich hier wegen ihrer schlichten Arithmetik auf Binärkodes beschränken) lassen sich grundsätzlich drei verschieden Lösungsansätze unterscheiden:

1. **Suchen in endlichen Mengen**, die allgemeinste Methode. Liegt ein beliebiger gültiger Kode vor, wird durch Probieren aller[17] möglichen Änderungen garantiert eine Lösung gefunden. Dabei braucht nicht in B gesucht werden, sondern nur in E, was durch $D_{\max}$ verschachtelte Schleifen leicht zu realisieren ist. Für einen Kode mit z. B. $k = 20$ muß pro Kodewort bis zu einer Million mal gehasht und mit dem Sollwert verglichen werden - machbar, aber aufwendig.

2. **Diskrete Optimierung** ist auf via Hashmatrix definierte Kodes beschräkt, versucht, das lineare Gleichungssystem $H\overrightarrow{b-x} = \overrightarrow{m}$ zu lösen unter Minimierung von $\overrightarrow{1} \cdot \overrightarrow{x}$. Verfängt sich die Optimierung in einem Zyklus oder lokalen Minimum mit $\overrightarrow{x} \notin E$, liefert diese Methode gar keine Lösung.

[17] statistisch gesehen findet man schon nach Absuchen der halben Menge eine Lösung

3. **Direktwahl** führt in einem bis $D_{\max}$ Schritten vom Syndrom, das ist die Differenz zwischen Ist- und Sollwert der Nachricht, zur kompletten Änderung. Dies ist die bevorzugte, aber bezüglich der Kodegenerierung auch die komplizierteste Variante.

4.2 Problem Schwund

Im Idealmodell eines steganographischen Systems sind Einbettung und Kodierung soweit entkoppelt, daß sie getrennt voneinander entwickelt werden können und weder Sicherheit noch Korrektheit des jeweils anderen stören[18]. Jede ausgelesene Nachricht sollte sich korrigieren und jede dazu nötige Änderung einbetten lassen. Wie bereits erwähnt, kann aber Schwund die geplante Einbettung vereiteln.

Es gibt mehrere Möglichkeiten, dem zu begegnen. Zum einen kann man das ermittelte Ergebnis komplett verwerfen und einen anderen Änderungsvektor suchen („Weitersuchen“), wozu allerdings $d_{\max}$ temporär inkrementiert werden muß[19] (die Hashfunktion h wird dabei nicht an das vergrößerte $d_{\max}$ angepaßt!). Dies funktioniert bei suchenden Algorithmen, hat aber folgenden Nachteil: auch die neuen Änderungen können zu Schwund führen - aufgrund des erhöhten $d_{\max}$ ist das sogar wahrscheinlich. Da zu manchen Vektoren (trivial: $\{\mathrm{DCTK}_1\}^n$) gar keine alternativen Änderungsvektoren existieren, ist diese Methode allein nicht brauchbar, kann aber als erster Versuch durchaus Anwendung finden. Wichtig ist nur, den unterdrückten Schwund irgendwo im Restbett zu erzwingen, um keine neuen Angriffspunkte für die Stegoanalyse zu liefern (beziehungsweise die Sicherheit beweisen zu müssen).

Zum anderen kann man den adressierten DCTK tatsächlich schwinden lassen und nimmt dafür einen neuen ins Kodewort auf. Diese Methode findet im Algorithmus **F5** Anwendung. Zwar kann sich auch dieses „Streichen und Nachrücken“ lange fortsetzen, aber spätestens, wenn alle DCTK_1 (original in Kodewort enthaltene wie nachgerückte) verschwunden sind, kann mit Sicherheit eingebettet werden. Bei der Parameterwahl muß berücksichtigt werden, daß diese Methode N verändert. Der sichere Weg lautet, hier die DCTK_1 gar nicht mitzuzählen, jedoch ist deren Anteil mit ca. 50 % [20]recht hoch, sodaß man sie teilweise doch mit einbezieht, sich dabei aber einem gewissen Risiko des Scheiterns[21] aussetzt. Die Wahrscheinlichkeit für das Auftreten von Schwund ist für jeden Einbettungsversuch nur von der maximalen Änderungsrate des Kodes $\frac{d_{max}}{n}$ und dem DCTK_1-Anteil am Bett p_{DCTK_1} abhängig, nämlich:

$$p_{\text{schwund}} = 1 - \left(1 - \tfrac{d_{max}}{n} p_{DCTK_1}\right)^n$$

Die maximale Anzahl N_{schwund} schwindender Bits beträgt statistisch also:

[18] unter Erfüllung zuvor gemachter Annahmen, zb Gleichverteilung von 0- und 1-Bits
[19] beim perfekten Kode gibt es genau einen gültigen Vektor pro Syndrom
[20] abhängig von Bildinhalt / Bildschärfe
[21] sollte das passieren, mit anderem Schlüssel probieren

$N_{\text{schwund}} = z \sum_{i>0} \left(p^i_{\text{schwund}}\right)$

Man benötigt einige wenige Iterationsrunden zur Berechnung, da sich Kode und Schwund gegenseitig beeinflussen. (In der Praxis spart man diesen Aufwand, indem man grob schätzt, daß die Hälfte der DCTK_1 schwindet.)

Kombiniert man beide Varianten und nimmt einigen zusätzlichen Aufwand in Kauf, bietet sich die Möglichkeit der „Schwundkodierung“. Da die Auswahl der Schwundstelle(n) prinzipiell beliebig ist, läßt sie sich natürlich auch so treffen, daß bei anschließender Kodierung möglichst wenige (eventuell gar keine) Änderungen nötig sind, was freilich nur nach Durchprobieren aller Möglichkeiten entschieden werden kann. Weiterhin ist ein Schwundkonto zu führen, das beim Streichen dekrementiert, beim Weitersuchen beibehalten und am Ende der Einbettung auf einen Stand von ungefähr 0 gebracht sein sollte. Initialisiert wird das Konto mit der für tatsächliche JPEG-Quantisierung erwarteten Anzahl für N_{schwund}. (Diese Variante berührt schon mehr die Einbettung als die Kodierung und ist nur zur Demonstration gedacht, daß Schwund keineswegs zu Problemen bei der Kodierung führen muß, sofern man vom Aufwand für die Berechnung letztlich nicht eingesetzter Änderungsvektoren absieht.)

Da es für das Auslesen keine Rolle spielt, mit welcher Variante der Schwund behandelt wurde, wird im folgenden nur noch jene des Streichens und Nachrückens betrachtet werden, ohne dadurch die anderen abwerten zu wollen.

4.3 Entwicklung von Algorithmen

Im folgenden werden verschieden Wege (und Irrwege) beschrieben, den passenden Änderungsvektor für ein Kodewort zu finden. Die Wahl eines Algorithmus bestimmt dann den Aufbau der Hashfunktion h und damit auch den Algorithmus zum Auslesen.

4.3.1 Suchen des Änderungsvektors

Mit dem Standardansatz „Monte Carlo“ (also schlichtem Durchprobieren) ist man in der Lage, immer einen passenden Änderungsvektor zu finden. Bezüglich seiner einfachen Struktur und der Eigenschaft, mit verschiedensten Kodeschemata umgehen zu können, scheint dieser Algorithmus optimal, jedoch ist der Aufwand untragbar, für jedes Kodewort 2^k mögliche Änderungen abzutesten. Deshalb sollen von diesem Algorithmus spezialisierte, aber schnelle Ableger entwickelt werden.

Schnellere Suche durch Aufspalten der Suchmenge Kann man sich a priori auf Teilmengen konzentrieren, läßt sich die Suche beschleunigen. Man stelle sich dazu folgende Aufgabe vor: Zerlegen eines Syndroms in zwei Vektoren einer n-elementigen Menge. Ist eine beliebig

gewählte Komponente (ein Bit) des Syndroms gleich 1, fallen sofort jene Vektorenpaare weg, die sich in dieser Komponente gleichen. Statt $\binom{n}{2} = \frac{n^2-n}{2}$ brauchen nur $\frac{n}{2}\frac{n}{2} = \frac{n^2}{4}$ Paare untersucht werden (symmetrische Aufteilung vorausgesetzt). Durch rekursives Zerlegen der Teilmengen anhand der übrigen Komponenten kann die Suche weiter beschleunigt werden, wobei die Komplexitätsklasse $\mathrm{O}(n^2)$ aber nicht verlassen werden kann.

Diese Methode stößt schnell auf Grenzen, z. B. schon bei Ausdehnung der Suche auf Vektorentripel (also bei größeren $d_{\max}$). Dies liegt an der „alles oder nichts"-Stuktur der Binärkomponenten. Das Hamminggewicht eignet sich hier besser, dafür muß die Hashmatrix aber eine ganz bestimmte Struktur aufweisen. Konstruktion und Vorgehensweise seien am $(3, 23, 11)$-Kode erläutert.

Einerseits muß natürlich ein gültiger Kode entstehen, andererseits soll das Syndromgewicht die Suchmenge möglichst stark begrenzen. Dies ist glücklicherweise kein Widerspruch. Man stelle sich hierzu das Gitter der Nachrichten vor, einen Hyperwürfel 11. Dimension, dessen Ecken die Information, dessen Kanten Änderungen eines Bits repräsentieren. Zwischen Ecke $(0,\ldots,0)^T$ und $(1,\ldots,1)^T$ liegen noch 10 parallele Ebenen mit aufsteigendem Hamminggewicht. Sie seien entsprechend mit Ebene_0 … Ebene_{11} bezeichnet.

Um die Ebenen möglichst gleichmäßig mit möglichst geringem Aufwand aufzuteilen, fügen wir dem impliziten Nullvektor der Hashmatrix noch den Vektor $(1,\ldots,1)^T$ hinzu, also jenen mit der größten Distanz zum Nullvektor. Damit kann zumindest grob kodiert werden, ob die Nachricht ein Gewicht kleiner 6 hat oder nicht. Das ist natürlich noch ungenügend, deshalb werden Vektoren hinzugefügt, die wiederum von allen bisherigen maximal entfernt sind. Das trifft auf alle Kodes der Grenzebene, also Ebene_6 zu. Dabei beschränkt man sich auf wenige, dafür gut verteilte Vektoren, wie sie durch eine paarweise Mindestdistanz von 6 gewährleistet ist. Damit sind bislang 13 Punkte/Nachrichten so verteilt, daß die Distanz zum Nachbarn immer 5 oder 6 beträgt (sofern k keine Zweierpotenz ist, verläuft die Aufteilung also nicht perfekt; unschön zwar, aber nicht schlimm). Im nächsten Schritt wären Vekoren mit paarweiser Mindestdistanz von 3 einzufügen, was aber gerade $d_{\max}$ entspricht. So genügt es, stattdessen die Einheitsmatrix hinzuzufügen, wodurch man sich um alle bisherigen Vektoren innerhalb eines Radius von 2, teilweise 3 frei bewegen kann. Tatsächlich wird so das gesamte Gitter abgedeckt. Eine mögliche Hashmatrix sieht so aus:

Syndromgewicht	zu durchsuchende Menge(n)
0	$\emptyset$
1	H_a
2	$H_a \times H_a$
3	$H_a \times H_a \times H_a$
4	$H_a \times H_a \times H_b$, $H_a \times H_b \times H_c$
5	$H_a \times H_b$, $H_b \times H_c$, $H_a \times H_b \times H_b$, $H_b \times H_b \times H_c$
6	H_b, $H_b \times H_b$, $H_b \times H_b \times H_b$, $H_a \times H_a \times H_b$, $H_a \times H_b \times H_c$
7	$H_a \times H_b$, $H_a \times H_b \times H_b$
8	$H_a \times H_a \times H_b$
9	$H_a \times H_a \times H_c$
10	$H_a \times H_c$
11	H_c

Tabelle 1: Teilmengen für Schnellsuche (Beispiel)

$$H = \underbrace{\begin{pmatrix} 1&0&0&0&0&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0&0&0&0 \\ 0&0&0&0&1&0&0&0&0&0&0 \\ 0&0&0&0&0&1&0&0&0&0&0 \\ 0&0&0&0&0&0&1&0&0&0&0 \\ 0&0&0&0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&0&0&0&0&1 \end{pmatrix}}_{H_a} \underbrace{\begin{pmatrix} 0&0&1&1&0&1&1&1&0&0&1 \\ 0&0&1&1&1&0&1&0&1&1&0 \\ 0&1&0&0&1&1&1&1&0&1&0 \\ 0&1&0&1&1&0&0&1&1&0&1 \\ 0&1&1&0&0&1&0&0&1&1&1 \\ 1&0&0&0&1&1&1&0&1&0&1 \\ 1&0&0&1&0&1&0&1&1&1&0 \\ 1&0&1&0&1&0&0&1&0&1&1 \\ 1&1&0&1&0&0&1&0&0&1&1 \\ 1&1&1&0&0&0&1&1&1&0&0 \\ 1&1&1&1&1&1&0&0&0&0&0 \end{pmatrix}}_{H_b} \underbrace{\begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}}_{H_c}$$

Dank dieser Kodierung muß nun nicht mehr in $H \times H \times H$ gesucht werden, sondern entsprechend Tabelle 1. (Eine solche Tabelle entsteht, indem man den Zeilen, die ein bestimmtes Gewicht repräsentieren, genau jene Elemente aus $\{\emptyset, H_a, H_b, H_c\}^3$ hinzufügt, durch die dieses Gewicht erreichbar ist.)

Der Worstcase tritt für ein Gewicht von 6 ein, wobei ca. 11^3 (statt der ursprünglichen ca. 23^3) Suchschritte nötig sind. Trotz allem, die Komplexitätsklasse von $O(n^{d\max})$ kann hierdurch nicht verlassen werden. Eine negative Eigenschaft dieser Methode ist, daß sich für nicht perfekte Kodes die Hashmatrix (bisher) nicht ohne menschliches Zutun generieren läßt. Somit ist die Anwendbarkeit generell fraglich, da man nur gelegentlich (und zufällig) auf nichtriviale perfekte Kodes treffen wird [7].

Schnellere Suche durch Aufspalten der Suchschritte Da der Aufwand mit $d_{\max}$ derart wächst, soll nun versucht werden, die Korrektur eines Kodewortes mit großem $d_{\max}$ auf die Korrektur mehererer Kodewörter mit geringerem $d_{\max}$ zurückzuführen.

Es sei anfangs ein $(1, n, k)$-Kode(-wort) gegeben, dessen Länge n verringert werden soll, was eine Vergrößerung von $d_{\max}$ voraussetzt. Dazu teilt man die Hashmatrix in eine linke und eine rechte Hälfte, wobei letztere entfernt werden soll und deshalb jeder ihrer Spaltenvektoren aus zwei Vektoren der linken Hälfte kombinierbar sein muß. (Die Teilung bei $\frac{n}{2}$ hier im Beispiel sei nur der Übersichtlichkeit halber verwendet. In der Realität schwankt sie zwischen $\sqrt{n}$ und n.)

Mit einer Wahrscheinlichkeit von $\frac{1}{2}$ entspricht das Syndrom einem Vektor der linken Hälfte, so daß nichts zu tun ist. Andernfalls muß es mit Aufwand $\mathrm{O}(n^2)$ zerlegt werden. Damit ist ein $(2, \frac{n}{2}, k)$-Kodewort entstanden, mit dem ebenso verfahren wird, wobei mit einer Wahrscheinlichkeit von $\frac{1}{4}$, $\frac{1}{2}$ bzw. $\frac{1}{4}$ keine, eine bzw. zwei Zerlegungen mit Aufwand $\mathrm{O}(n^2)$ nötig sind. So entsteht aus einem anfänglichen (d, n, k)-Kode nach t Durchläufen ein $(2^t d, n', k)$-Kode. Für alle Kodes, deren $d_{\max}$ eine Potenz von 2 ist, beträgt die Komplexität einer Korrektur $\mathrm{O}(d_{\max} n^2)$. Damit ist es tatsächlich gelungen, die Komplexität für einige Kodes mit $d_{\max} > 2$ zu reduzieren.

Natürlich wird eine spezielle, erweiterte Hashmatrix benötigt. Deren Aufbau ist eigentlich trivial, läßt sie sich doch aus jeder gegebenen Originalmatrix direkt über die Forderungen errechnen: für jeden Durchlauf werden all jene neuen Vektoren zur rechten Seite der Matrix hinzugefügt, die sich aus Paaren der linken Seite kombinieren lassen. Hinderlich ist allerdings die enorme Größe der Matrix von 2^k Spalten[22]. (Für den doppelten Speicherplatz könnte man schon zu jedem Syndrom die beiden Indizes seiner Zerlegung abspeichern!)

Paarweise Zerlegung mittels Partnerwahl Da obiges Verfahren die erste spürbare Verbesserung darstellt, soll versucht werden, den aufwendigsten Teil davon weiter zu optimieren. Man könnte in der Tat die Suche über ca. n^2 Vektorpaare sparen, falls es ein leicht berechenbares Prädikat gibt, das für jeden Vektor anzeigt, ob er im linken oder rechten Teil[23] der Hashmatrix liegt. Dann ist nur noch zu jedem Vektor der linken Seite dessen Partner zu suchen - das ist einfach der Differenzvektor zwischen dem Syndrom und ihm selbst - und anhand des Prädikates auf Zugehörigkeit zur linken Seite zu testen. Dies liefert eine Zerlegung mit linearem Aufwand über n. Leider konnte ein solches Prädikat noch nicht für den Durchlauf durch alle Stufen gefunden werden (nicht einmal beispielhaft), denn hier stehen sich die Forderungen nach einem gültigen Kode überhaupt, der kompakten Speicherbarkeit der Matrix und der Prädizierbarkeit der Seitenzugehörigkeit gegenüber. Können die Probleme (außer-

[22] Hier wäre eine Automat (z B. LFSR) zur virtuellen Speicherung hilfreich, der in passender Reihenfolge einen Zyklus durch alle Spalten liefert.

[23] entsprechend der gerade aktuellen Teilung

halb dieser Arbeit) doch noch gelöst werden, steht ein Korrekturalgorithmus der Komplexität $O(d_{\max} n)$ zur Verfügung.

4.3.2 Anpirschen an den Änderungsvektor

Im Gegensatz zum normalen Suchen, bei dem die Reihenfolge der Suchschritte beliebig ist, da diese unabhängig voneinander sind, soll nun versucht werden, sich anhand der Ergebnisse vorheriger Suchschritte dem Änderungsvektor möglichst zielstrebig zu nähern. Leider greifen die aus der Mathematik bekannten Optimierungsverfahren[24] hier nicht, da es sich aufgrund der Modulo-2-Arithmetik nicht mehr um ein lineares Problem handelt. Die Umformung in ein solches ist mit erheblichem Aufwand verbunden (u.a. Einführung von Hilfs-Ungleichungen für jedes Bit) und verspricht keine effiziente Lösung. Stattdessen soll eine andere, einfache Optimierungsmethode diskutiert werden.

Korrektur nach maximaler Ähnlichkeit Hierbei wird versucht, das Syndrom schrittweise (natürlich möglichst schnell, dh. in höchstens $d_{\max}$ Schritten) zu $\overrightarrow{0}$ zu minimieren. Intuitiv geschieht das durch Subtraktion jenes Hashvektors mit der größten Übereinstimmung zum Syndrom (via Negation des entsprechenden Bettbits). Nach Neuberechnung des Syndroms (via Addition des Hashvektors) ist das solange zu wiederholen, bis das Syndrom zu $\overrightarrow{0}$ wird. Da es mehrere „ähnlichste“ Hashvektoren[25] geben kann, wird durch Backtracking der ganze Baum abgearbeitet. Trotzdem gibt es Konstellationen (und derer gar nicht so wenige) ohne Lösung.

Verläßt man nun temporär die Modulo-2-Arithmetik, erhält man bei der Matrixmultiplikation einen k-stelligen Vektor natürlicher Zahlen und damit ein zusätzliches Kriterium – eine Wichtung der Stellen des Syndroms. Weiterhin lassen sich die Komponenten der Hashmatrix oder des Bettes mittels $f : x \to (-1)^x$ transformieren. Trotz der Zusatzinformation ist noch keine schnelle Korrektur möglich, schlimmer noch: Listet man für einige Nachrichten das Matrixprodukt, die Änderungsstellen und das (daraus rückentwickelte) Syndrom, scheint es keinen erkennbaren Zusammenhang zwischen den Größen zu geben, wenn man davon absieht, daß natürlich die Summe der gewählten Spaltenvektoren der Hashmatrix das Syndrom ergibt.

Hashen und Korrektur über Netzstrukturen Da das Matrixprodukt offenbar nicht genügend Information zur Korrektur bereitstellt, sind Strukturen zu suchen, die mehr über den Zusammenhang zwischen den Bits verraten. Von Interesse sind hier die temporären Daten, die während der Matrixmultiplikation entstehen. Dabei handelt es sich jedoch um schwer interpretierbare, abstrakte Zahlen, weshalb hier zu einer anderen Beschreibungsform, nämlich

[24] z. B. Simplexverfahren

[25] aufgrund gleicher Hammingdistanz zum Syndrom

mehrstufigen Netzwerken übergegangen werden soll. Grob gesagt ist die Hashfunktion oft so aufgebaut, daß jedes Bit aus Bett bzw. Nachricht mit ca. der Hälfte der Bits der jeweils anderen Menge verbunden ist. Da sich die Pfade zwischen verschiedenen Bits teilweise überschneiden, ist die Hoffnung, ausgehend vom Syndom Fehlerstellen beim Rückpropagieren so geschickt in gemeinsame Bahnen lenken zu können, daß höchstens $d_{\max}$ Korrekturstellen übrigbleiben. Ein solches Netzwerk läßt sich zu jeder Hashmatrix erzeugen, indem diese entsprechend stückweiser Übereinstimmung ihrer Spaltenvektoren zerlegt wird[26]. Etwas anschaulicher formuliert heißt das, wiederholt gemeinsame Terme auszuklammern und in Zwischenschichten einzutragen.

Ausgehend vom Syndrom beschickt man die Netzknoten mit "1" für "Änderung nötig", mit "-1" für "beibehalten" und mit "0" für "beliebig/unentschieden". Stoßen Pfade aufeinander, werden die Werte addiert, gabelt sich der Weg, muß der Wert aufgeteilt werden, entweder entsprechend einer Präferenz oder symmetrisch. Je größer die Knotenbewertung, desto notwendiger eine Änderung des Knotens (das betrifft hauptsächlich die Eingangsschicht, also das Bett). Soweit entstehen Ergebnisse entsprechend dem vorigen Abschnitt, es gibt aber zwei Vorteile. Erstens läßt sich so gut Schwund unterdrücken (durch Vorwärtspropagation der Abhänigkeit von DCTK_1 werden anschließend die Entscheidungen beim Rückpropagieren beeinflußt), zweitens können Fehlentscheidungen leichter rückgängig gemacht werden, indem nur eine oder einige wenige Stufen im Netz neu zu bewerten sind.

Recht kompakt läßt sich ein solches Netz mit der Methode der Schnelltransformation beschreiben und berechnen (es gibt keine Abkürzungen und pro Netzschicht (=Transformationsschritt) n Knoten, die stets mit je 2 Knoten der beiden Nachbarschichten verbunden sind), Dabei wird die Hashfunktion durch diese Transformation bestimmt, und damit auch $d_{\max}$ (nicht unbedingt optimal).

4.3.3 Direktwahl des Änderungsvektors

Hier soll ein kleines, leider nicht funktionierendes Beispiel die erhoffte Funktionsweise demonstrieren: nachdem auf die übliche Weise das Syndrom bestimmt und mit 0 verglichen wurde, wird dieses als Zahl interpretiert und durch n dividiert. Der Divisionsrest entspricht dem Index[27] eines der zu ändernden Bettbits, und zwar derart, daß nach höchstens $d_{\max}$ Wiederholungen der Prozedur die Korrektur vollständig ist.

Hashen und Korrektur über Kodebäume Der Nachteil der unter 4.3.2 beschriebenen Methode ist, daß beim Rückpropagieren Fehlentscheidungen möglich sind, die sich fortpflanzen, und erst am Ende erkennbar ist, ob überhaupt Fehlentscheidungen getroffen wurden.

[26] diese Zerlegung muß/wird nicht eindeutig sein
[27] im aktuellen Kodewort

Deshalb ist die Netzstruktur durch eine Baumstruktur zu ersetzen, deren Blätter die Bettbits enthalten und deren Wurzel die komplette Nachricht bestimmt. Damit ist jeder Knoten nur noch direkter Unterknoten genau eines anderen Knotens. Die Bewertung der Knoten ist nun nicht mehr binär, also maximal 1, sondern maximal $S(n_i, d_{\max}) - 1$. Die Größe n_i gibt die Anzahl der transitiv zugehörigen Blätter für den Knoten i wieder. Zu jedem Knoten ist eine Tabelle zu generieren mit den Einträgen $\{0/0,\ 1/0,\ 0/1,\ 2/0,\ 1/1,\ 0/2,\ 3/0,\ 2/1,\ \ldots\}$, die den Anzahlen der Änderungen im linken und rechten Unterknoten entsprechen und denen die Kombinationen der Bewertungen dieser Knoten sowie die daraus resultierende Bewertung des aktuellen Knotens zugeordet werden. Damit ist sowohl das Hashen definiert als auch die Korrektur, denn zu jedem (Teil-)Syndrom des Knotens steht fest, auf welche Seite wieviele Änderungen zu delegieren sind. Es taucht jedoch folgendes Problem auf: die Anzahl der möglichen Bewertungen ist nur sehr selten ein Teiler des Produktes der Anzahlen von Bewertungen der Unterknoten - somit ist keine gleichmäßige Zuordnung möglich. Unter anderem dadurch entsteht die merkwürdige Situation, daß teilweise eine Korrekturkomplexität von $\mathrm{O}(d_{\max} \log(n))$ erreicht werden kann, sonst aber gar keine Lösung gefunden wird. In einem solchen Falle ist der normale Suchalgorithmus einzusetzen, wobei die Komplexität des Hashens $\mathrm{O}(n \log(n))$ beträgt und beim Ändern $d_{\max}$ nicht immer eingehalten werden kann.

Korrektur durch Numerierung der Änderungsvektoren Sofern man die Hashfunktion vorerst undefiniert läßt, kann man sich ganz auf die Korrektur konzentrieren und eine möglichst einfache Abbildung h^{-1} festlegen. Dazu bietet es sich an, die Änderungsvektoren zu sortieren[28] und zu numerieren, wobei die vergebenen Nummern den Nachrichten entsprechen. Über wenige Zugriffe auf sehr kleine Tabellen kann so der komplette Änderungsvektor anhand des Syndroms zusammengesetzt werden. Dies stellt aber keinen Gewinn dar, denn der aufwendige Teil wurde nur zum Hashen und zur Syndrombestimmung verlagert.

5 Auswertung

Leider konnte kein schneller Algorithmus für Kodes mit größerem $d_{\max}$ gefunden werden, der sich in einer Implementierung hätte niederschlagen können. Die Mischkodes mit $d_{\max} = 1$ dagegen wurden in das Programmpaket F5 (`http://wwwrn.inf.tu-dresden.de/~westfeld/f5`) integriert und getestet. Die theoretisch mögliche Verbesserung liegt bei maximal einem zusätzlich einbettbaren Bit pro Änderung, was durch die Effizienz der $(1, n, k)$-Kodes von $W_k = \frac{K}{D} = \frac{2^k}{2^k-1}k \approx k$ bedingt ist und beim Mischen nur Kodes mit $k_0 = k$ und $k_1 = k + 1$ Verwendung finden. Dies bedeutet eine maximale Steigerung um den Faktor $\left(\left(1 + \frac{1}{k}\right)\left(1 - \frac{1}{2^{k+1}-1}\right) - 1\right)$; durchschnittlich wird in der Praxis nur die Hälfte davon erreichbar sein. Tabelle 2 zeigt die Erwartungswerte der Verbesserung für die von F5 unterstützten

[28] z. B. nach Größe oder nach Gewicht.

k	1	2	3	4	5	6
Verbesserung der Effizienz W	+17 %	+14 %	+12 %	+10 %	+9 %	+8 %

Tabelle 2: Erwartete Steigerung der Einbettungseffizienz durch Mischkodes

Kodes. Somit kann man beispielsweise gegenüber einem reinen Kode mit Parameter $k = 3$ eine um 12 % größere Nachricht in das gleiche Bild einbetten, ohne daß die Anzahl an Änderungen erhöht wird.

Tritt beim Einbetten Schwund auf, verringern die zusätzlich gemachten Änderungen die Effizienz:

$$W_{k,p_{DCTK1}} = \frac{1 - p_{DCTK1}}{1 - p_{DCTK1}^{1+Np_{DCTK1}}} W_k$$

Dies betrifft reine wie gemischte Kodes gleichermaßen, weshalb die relative Verbesserung davon nicht beeinflußt wird, die Erwartung für die absolute Verbesserung dagegen schrumpft auf ca. $\frac{1}{4}$Bit pro Änderung (Annahme: $p_{DCTK1} \approx \frac{1}{2}$).

Tabelle 3 zeigt für einige ausgewählte Hüllen und Nachrichten die erreichte Einbettungseffizienz W. Durchgeführt wurde die Messung mit für Mischkodes günstigen Parametern N und K, mit je drei zufällig erzeugten Passwörtern. (Der für Mischkodes ungünstigste Fall ist uninteressant, da sich dann reine Kodes und Mischkodes gleichen.[29])

Weiterhin wurde die Effizienz beim Einbetten einer zufällig erzeugten Nachricht in ein zufällig gewähltes Bild (wobei 50 Bilder zur Wahl standen) gemessen und gemeinsam mit dem verwendeten Kodeparameter k aufgezeichenet. Die Messung wurde beendet, als für jedes k mindestens 150 Messwerte vorlagen. Der durchschnittliche DCTK_1-Anteil betrug dabei 41 %. Tabelle 4 zeigt die Ergebnisse im arithmetischen Mittel. Sie entsprechen größtenteils den Erwartungen. Für Mischkodes mit $k = 5$ sind die Daten durch Diagramm 10 ausführlich dargestellt. Wie erwartet liegen die meisten Punkte zwischen den Kodegrenzen der Kodes mit $k = 4,5$ und $k = 5,6$.

6 Zusammenfassung

Das Ziel der Arbeit konnte teilweise erreicht werden. So steht mit den entwickelten Mischkodes eine leistungsfähige Kodeklasse zur Verfügung, die es erlaubt, mit geringem Aufwand die Bettkapazität nahezu optimal auszunutzen. Der Umstieg auf die informationstheoretisch

[29] Natürlich streuen die Ergebnisse, man denke an die Extremfälle „es muß zufälligerweise nichts geändert werden“ und „es muß immer geändert werden und alle DCTK_1 schwinden “

Bett (davon $DCTK_1$) [Bits]	Nachricht [Bytes]	k [Bits]	W_{F5} [Bits/Änderung]	$W_{Mischkode}$ [Bits/Änderung]	Verbesserung [Bits/Änderung]
14321 (7176)	888	1	1.4, 1.4, 1.5	1.7, 1.7, 1.7	0.3
	570	2	1.7, 1.7, 1.6	2.0, 1.9, 2.0	0.3
	345	3	2.0, 2.0, 2.0	2.3, 2.3, 2.3	0.3
	213	4	2.3, 2.2, 2.3	2.6, 2.7, 2.5	0.3
	126	5	2.6, 2.6, 2.8	3.0, 3.0, 3.1	0.4
	63	6	2.6, 3.2, 3.1	2.8, 3.4, 3.1	0.1
101842 (1612)	8420	1	1.9, 1.9, 1.9	2.6, 2.6, 2.6	0.7
	5415	2	2.6, 2.6, 2.6	3.3, 3.3, 3.3	0.7
	3366	3	3.3, 3.3, 3.3	4.2, 4.2, 4.1	0.9
	2035	4	4.2, 4.2, 4.1	5.0, 5.0, 5.0	0.8
	1122	5	5.0, 5.0, 5.0	5.9, 5.9, 5.8	0.9
	690	6	5.8, 5.9, 5.8	6.7, 6.7, 6.8	0.9
117771 (62801)	7145	1	1.4, 1.4, 1.4	1.6, 1.6, 1.6	0.2
	4595	2	1.7, 1.7, 1.7	1.9, 1.9, 1.9	0.2
	2860	3	2.0, 1.9, 1.9	2.3, 2.2, 2.2	0.3
	1730	4	2.2, 2.3, 2.3	2.6, 2.6, 2.6	0.3
	1015	5	2.6, 2.6, 2.7	3.1, 2.9, 3.1	0.4
	590	6	3.1, 2.9, 2.9	3.2, 3.5, 3.4	0.4

Tabelle 3: Vergleich reiner und gemischter Kodes (gemessen)

k	1	2	3	4	5	6
W (reine Kodes)	1.1	1.6	2.1	2.5	3.1	3.6
W (Mischkodes)	1.3	1.8	2.4	2.8	3.4	3.9
Steigerung der Effizienz W	+18 %	+13 %	+14 %	+12 %	+10 %	+8 %

Tabelle 4: Gemessene Steigerung der Einbettungseffizienz durch Mischkodes

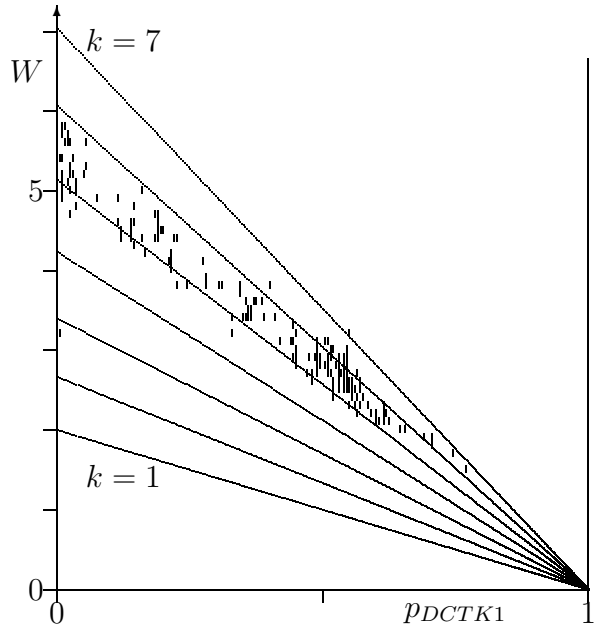


Abbildung 10: Messung der Einbettungseffizienz W für Mischkodes mit $k = 5$

günstigeren Kodes mit einem $d_{\max} > 1$ konnte dagegen nicht geschafft werden, da die Kodierung trotz einiger Verbesserungen noch zu aufwendig ist. Die vorgestellten Ansätze lassen aber hoffen, daß es dennoch möglich ist, mit einer Komplexität von $\mathrm{O}(d_{\max} n)$ zu korrigieren.

Literatur

[1] R. Crandall: „Some notes on steganography“, 1998,
`os.inf.tu-dresden.de/~westfeld/crandall.pdf`

[2] R.W. Hamming: „Error detecting and error correcting codes“, Bell Syst. Tech. J. 29, 1950

[3] H. Klimant, R. Piotraschke, D. Schönfeld: „Informations- und Kodierungstheorie“, Teubner Verlagsgesellschaft Stuttgart Leipzig, 1996

[4] A. Westfeld: „Steganographie in komprimierten Videosignalen“, Diplomarbeit, Technische Universität Dresden, 1997

[5] A. Westfeld: „F5 - Ein Steganographischer Algorithmus“, 1999,
`os.inf.tu-dresden.de/~westfeld/publikationen/vis01.pdf`

[6] J. Zeidler: „Untersuchung des steganographischen Algorithmus F4“, Diplomarbeit, Technische Universität Dresden, 2000

[7] V.A. Zinovjev, V.K. Leontjev: „On perfect codes“, 1972