Digitale Datenverarbeitung im Amateurfunk – wie die Bits laufen

Inhaltsverzeichnis

1. Vorwort

Über lange Zeit beschränkte sich die „Digitale“ Datenübertragung im Amateurfunk auf CW und RTTY. Die Verfügbarkeit von Mikroprozessoren und PC erweiterte die Möglichkeiten der digitalen Datenübertragung gewaltig. Inzwischen haben wir eine wahre Inflation von Datenübertragungsverfahren mit teilweise verblüffenden Fähigkeiten der Fehlerkorrektur. Eine Reihe dieser Verfahren habe ich in der Vergangenheit ganz oder teilweise „nachgebaut“ um die Mechanismen zu verstehen die hinter diesen Verfahren stecken. Dabei hat sich gezeigt, dass es so etwas wie einen Baukasten von Verfahren gibt, aus dem je nach Anwendungsfall (z.B. Kurzwellenbetrieb oder UKW-Betrieb, hoher Datendurchsatz oder sichere Übertragung) verschiedene dieser Bausteine zum fertigen Übertragungsverfahren zusammengesetzt werden. Über diese Bausteine (Verfahren) soll in diesem Vortrag berichtet werden.

2. Die Verfahren

In dem Baukasten für die Entwicklung solcher Verfahren gibt es eine Reihe von verschiedenen Schubladen:

- Fehlererkennung
- Fehlerkorrektur
- Verschlüsselung
- Verwürfelung
- Datenkomprimierung
- Synchronisation
- Modulation
- Digitale Filterung

In jeder dieser Schubladen findet sich eine Reihe von verschiedenen Bausteinen mit deren Hilfe eben diese Verfahren realisiert werden können. Natürlich gibt es auch Abhängigkeiten die dabei beachtet werden müssen. So macht es zum Beispiel keinen Sinn eine Fehlerkorrektur einzubauen, wenn man die Fehlererkennung vergessen hat.

In den folgenden Kapiteln sollen die verschiedenen Schubladen und deren Bausteine genauer beschrieben werden.

2.1 Die Fehlererkennung

Bei den ersten digitalen Verfahren im Amateurfunk gab es noch keine Fehlererkennung. Bei CW kann man einen Fehler nur erkennen, wenn man ein Zeichen empfängt dass es nicht gibt. Bei allen anderen lässt sich auf einen Fehler höchstens im Textzusammenhang schließen. Die

Fehlererkennung in einer Art „übergeordneter Schicht“ - wie dem Textzusammenhang ist in der digitalen Datenübertragung nicht unüblich. Bei unserem speziellen Fall CW lässt sich die Fehlererkennung dadurch realisieren, dass man wichtige Passagen wiederholt. Unterscheidet sich eine empfangene Nachricht von deren Wiederholung, ist klar, dass eine von beiden Nachrichten mindestens einen Fehler haben muss.

Das zweite digitale Verfahren im Amateurfunk – das RTTY hat dasselbe Problem. Bei RTTY kommt außerdem noch hinzu, das es auch Abhängigkeiten innerhalb eines Textes gibt die durch die Buchstaben-Ziffern-Umschaltung bedingt sind. Wurde z.B. eine Umschaltung von Buchstaben auf Ziffern nicht korrekt übertragen, sind alle folgenden Zeichen bis zur nächsten Ziffern-Buchstaben-Umschaltung falsch. Auch hier lässt sich auf übergeordneter Ebene – also im Text – der Fehler oft lokalisieren. Dieser Weg ist aber außerhalb des eigentlichen RTTY-Verfahrens. Um eine Fehlererkennung zu ermöglichen wiederholt man auch in RTTY wichtige Passagen wie bei CW.

Um Übertragungsfehler im Verfahren selbst zu erkennen ohne die empfangene Nachricht interpretieren zu müssen, muss jedem Zeichen eine Zusatzinformation mitgegeben werden. Eine sehr früh angewendete Methode ist die des Paritybits.

2.1.1 Das Paritybit

Während sich beim Amateurfunk in RTTY der 5-Bit lange Baudotcode durchgesetzt hat, hat sich in der Computerwelt der ASCII-Code durchgesetzt. Dieser Code verwendet 7 oder 8 Bits für die Zeichencodierung. Soll zusätzlich eine Fehlererkennung eingebaut werden, wird an jedes Zeichen ein weiteres Bit angehängt, das Partiybit. Dieses Bit wird so gesetzt, dass die Summe aller übertragenen „1“ in dem betroffenen Zeichen gerade ist (gerade Parity) oder ungerade (ungerade Parity). Der Empfänger kann jetzt in jedem empfangenen Zeichen die Anzahlen der „1“ ermitteln und feststellen ob sie der Parityvorgabe entsprechen. Stimmt die Anzahl nicht, so gab es in dem Zeichen mindestens einen Übertragungsfehler. Leider ist auch dieses Verfahren nicht wirklich sicher denn zwei Fehler innerhalb eines Zeichens werden nicht als Fehler erkannt. Eine genauere Untersuchung dieses Problems führt zu der Definition des Hamming-Abstandes der die Fehlerempfindlichkeit einer digitalen Kodierung beschreibt.

2.1.2 Der Hammingabstand

Zur Ermittlung des Hammingabstandes wird die Anzahl der Übertragungsfehler gezählt bis anstatt des korrekten Zeichens ein anderes dekodiert wird ohne dass ein Fehler erkannt wird. Beim oben beschriebenen Paritybit-Verfahren wäre der Hammingabstand damit „2“:

- Ein Fehler wird erkannt (aus einer geraden Anzahl wird eine ungerade und umgekehrt)
- Ein zweiter Fehler führt zu einem anderen Zeichen, die Parityinformation scheint zu stimmen, die **beiden** Übertragungsfehler werden nicht erkannt.

Entsprechend hätte RTTY dann den Hammingabstand 1: Jeder Übertragungsfehler führt zu einem falschen Zeichen. Ein Übertragungsfehler kann also nicht erkannt werden.

Es gibt nun eine Reihe von Codes bei denen nicht alle Zeichen zu ihren nächsten Nachbarn denselben Hammig-Abstand haben. Wichtig für die Bewertung einer Kodierung ist immer der kleinste Abstand der innerhalb des gesamten Codes auftritt.

Man kann nun Codes bauen bei denen jedes Zeichen mindestens drei Änderungen braucht damit ein anderes Zeichen daraus wird. (Hamming-Abstand 3). Bei einer solchen Kodierung ist es möglich

einen oder zwei Übertragungsfehler innerhalb eines Zeichens zu erkennen. Geht man davon aus, dass es nur einen Übertragungsfehler gegeben hat, lässt sich das ursprüngliche Zeichen sogar wieder rekonstruieren. Haben sich in der Übertragung statt des vermuteten einen Fehlers zwei eingeschlichen, dann führt die „Rekonstruktion“ zu einem fehlerhaften Ergebnis. Drei Übertragungsfehler innerhalb eines Zeichens werden nicht erkannt sondern das fehlerhaft empfangene Zeichen als scheinbar korrekt dekodiert.

2.1.3 Fehlererkennung am Beispiel AMTOR

AMTOR ist im Prinzip das in den Amateurfunk übernommene SITOR Verfahren. Grund für die Entwicklung des SITOR Verfahrens war die Notwendigkeit im kommerziellen Bereich eine sichere Datenübertragung zur Verfügung zu haben. SITOR gibt es in zwei Varianten – eine die eine 1 zu N Kommunikation erlaubt und eine die für eine 1 zu 1 Kommunikation gedacht ist. Die Unterschiede zwischen den beiden Übertragungsverfahren werden im Punkt Fehlerkorrektur genauer behandelt.

Die Zeichen sind so kodiert:

A	1000111
B	1110010
C	0011101
D	1010011
E	1010110
F	0011011
G	0110101

Tabelle 1: Amtor - Kodierung der ersten 7 Zeichen des Alphabets

Alle Zeichen bestehen also aus 3 Nullen und 4 Einsen. Hier ist der Hammingabstand 2 weil es Fälle gibt bei denen mit zwei Änderungen ein anderes Zeichen entsteht (aus einer der Einsen wird eine Null, aus einer der Nullen wird eine Eins). Natürlich gibt es auch eine ganze Anzahl von Fehlern die zu keinem anderen gültigen Zeichen führen. Damit werden bei dieser Art der Kodierung im Mittel mehr Fehler erkannt als es der minimale Hammingabstand von 2 vermuten ließe.

Eine genauere Untersuchung zeigt, dass 35 gültigen Zeichen 93 ungültige gegenüberstehen.

2.1.4 Das CRC Verfahren

Will man „nur“ wissen ob eine ganze Botschaft (also nicht nur ein Einzelzeichen) korrekt übertragen wurde kann man das mit einer „Prüfsumme“ verifizieren, die an das jeweilige Ende der Botschaft angehängt wird. Diese Prüfsumme wird oft nach dem CRC (cyclic redundancy check) Verfahren ermittelt. Je nach der Sicherheit mit der man die Aussage haben möchte gibt es diese Prüfsumme in verschiedenen Längen. Wie die Prüfzahl ermittelt wird wird meist durch das zugehörige Generatorpolynom beschrieben. So sieht das Generatorpolynom aus das bei der USB-Übertragung benutzt wird: $G = x^5 + x^2 + 1$ Zu dieser mathematischen Darstellung gibt es eine einfache, mit Logikbausteinen aufbaubare Struktur. In unseren Fall sieht diese Struktur so aus:

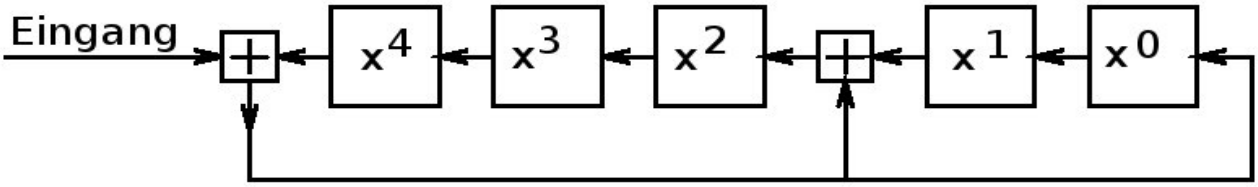

Bild1: Aufbau der CRC-Berechnung für das Generatorpolynom $G=x^5+x^2+1$ mit Logikelementen

Dabei sind die mit x^n bezeichneten Kästchen „ein Bit Schieberegister“. Das „+“ steht hier auch nicht für eine Addition sondern für ein Exklusiv-Oder. Sind die Nutzdaten alle „eingetaktet“ worden, steht in den Zellen x^0 bis x^4 die Prüfsumme.

Letztendlich entspricht die CRC-Operation einer binären Polynomdivision der Nachricht durch das Generatorpolynom. Das Ergebnis ist der Divisionsrest. Je größer das Polynom ist durch das dividiert wird, desto größer kann natürlich der Divisionsrest werden. In unserem Beispiel gäbe es bei 5 Ergebnisbits $2^5=32$ mögliche Divisionsreste . Um Verwirrungen zu vermeiden – die Rechenregeln der binären Polynomdivision unterscheiden sich etwas von denen, die wir von „normalen“ Zahlen her kennen.

Um ein Gefühl dafür zu erhalten was sich mit dem CRC Verfahren an Fehlern erkennen lässt nehmen wir an wir würden eine 16 Bit lange Zahl mit dem Generatorpolynom aus unserem Beispiel sichern, dann würden von 65536 möglichen Übertragungsfehlern $2^{16}/32=2048$ nicht erkannt. Zugegeben – die Aussage ob die Originalnachricht richtig oder falsch übertragen wurde ist in dem Fall nicht gerade sicher. Ein größeres Generatorpolynom oder ein kleinerer Nutzdatenblock würde das Ganze natürlich etwas weiter in Richtung „mehr Sicherheit“ schieben.

Obwohl das CRC-Verfahren keine Antwort darauf liefert welche Bits in dem übertragenen Block fehlerhaft sind, kann es im Zusammenspiel mit Kodierungen mit größerem Hammingabstand auch zur Rekonstruktion von Daten verwendet werden (wie es z.B. bei der Betriebsart OPERA der Fall ist).

2.1.5 Reed-Solomon Verfahren

Bei diesem Verfahren wird die gesamte Nachricht in kleinere Blöcke zerlegt und an jeden Block eine Information angehängt. Diese Sicherungsinformation besteht aus einer Fouriertransformation der Nutzdaten. Mit dieser Sicherungsinformation kann man nicht nur erkennen, ob die Übertragung fehlerfrei war, man kann Übertragungsfehler auch gleich korrigieren. Ein weiterer Vorteil dieses Verfahrens ist, dass man damit auch „Bündelfehler“ - also Fehler die sich über mehrere hintereinanderliegende Bits erstrecken – korrigieren kann. Dieses Verfahren wird im Amateurfunk bei der Betriebsart JT85 benutzt.

2.1.6 Faltungscodes

Anstatt Folgen von Nachrichten und Sicherungsinformationen in sogenannten Blockcodes hintereinander zu übertragen gibt es noch die Möglichkeit beides kombiniert zu übertragen. Diese Mischung von Nutz- und Redundanzinformation heißt Faltungscode. Dabei wird die zu übertragende Information bitweise in ein Schieberegister geschoben. An verschiedenen Stellen

werden die Zelleninhalte abgegriffen und per XOR verknüpft. Auf diese Weise werden aus der Eingangsbitfolge mehrere Ausgangsfolgen erzeugt, die zu „Symbolen“ zusammengefasst werden. In dem Kodierungsergebnis ist die ursprüngliche Information nicht mehr direkt zu erkennen. Dieses Verfahren wird bei den im Amateurfunk üblichen Übertragungsverfahren PSK31 und WSPR verwendet.

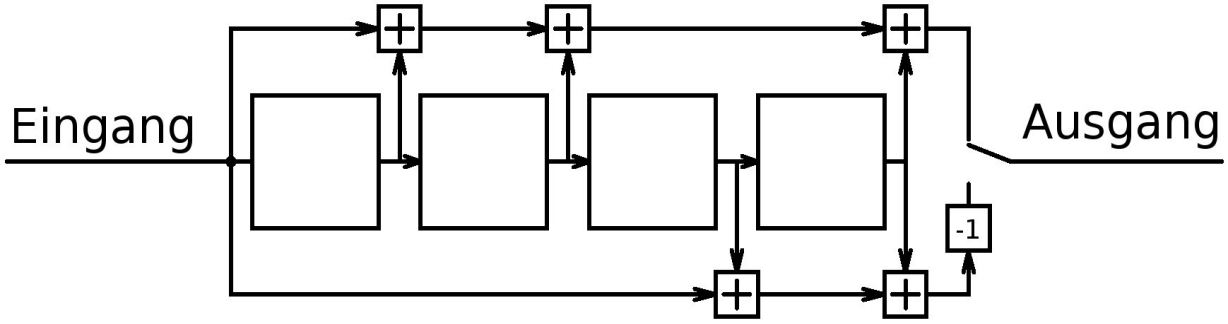

Bild 2: Faltungskodierer wie er bei PSK31 (QPSK-Mode) verwendet wird. Die „+“-Zeichen sind XORs, das Kästchen mit „-1“ ist ein Invertierer, die leeren Kästchen sind Schieberegisterzellen.

Faltungscodes wurden 1955 zum ersten mal beschrieben. Ein optimales Verfahren um die Information wieder herstellen zu können wurde allerdings erst 1967 von Herrn Viterbi vorgeschlagen. Das Verfahren wird deshalb als Viterbi-Dekoder bezeichnet. Faltungscodes haben den Vorteil, dass die **Kodierung** sehr einfach ist und auf beliebig lange Datenströme angewendet werden kann. Nachdem hier der Hauptaufwand bei der Entschlüsselung dieser Informationen liegt, soll der Viterbi-Dekoder in einem eigenen Punkt genauer beleuchtet werden.

3. Fehlerkorrektur

Die Information dass Nachrichten fehlerhaft übertragen wurden ist für sich alleine nicht allzu hilfreich. Deshalb wird dem Fehlererkennungsverfahren meist noch ein Fehlerkorrekturverfahren nachgeschaltet. Dabei gibt es zwei grundsätzlich unterschiedliche Verfahren:

- Die Wiederholung der fehlerhaften Nachricht
- Die Restaurierung mit Hilfe der mitgeschickten Sicherungsinformation.

Die „Wiederholung der fehlerhaften Nachricht“ setzt voraus, dass der Empfänger dem Sender mitteilen kann, dass bei der Übertragung ein Fehler aufgetreten ist. Das geht natürlich nur im 1:1-Betrieb. Ein CQ-Ruf oder im Kommerziellen die Aussendung eines Wetterberichts lässt sich mit diesem Verfahren nicht machen. Außerdem muss man bei diesem Verfahren bedenken, dass Übertragungsfehler an allen Stellen (in der ursprünglichen Nachricht selbst, in der Aufforderung zur Wiederholung oder in der Wiederholung selbst) auftreten können. Um in solchen Fällen Endlosschleifen oder blockierte Sender oder Empfänger zu vermeiden muss es für alle Fälle genau definierte Verhaltensweisen geben, die zu einem sogenannten „Übertragungsprotokoll“ zusammengefasst werden. Solche Verfahren sind im Amateurfunk im AMTOR-B, PACTOR oder PACKET realisiert.

Im Amateurfunk sind mit Abstand die meisten digitalen Verfahren solche die 1:N – Betrieb erlauben (ein Sender, viele Empfänger). Deshalb soll auch hier das Hauptgewicht auf den 1:N-Verfahren liegen.

3.1 Fehlerkorrektur bei CW und RTTY

Wie ganz am Anfang schon beschrieben – bei diesen Verfahren gibt es keine Fehlerkorrektur – zumindest keine, die das Übertragungsverfahren selbst bietet. Die Sicherheit der Übertragung lässt sich nur erhöhen in dem man selbst die Informationen die einem wichtig erscheinen vorsorglich mehrfach sendet (z.B. QTH und Namen).

3.2 Fehlerkorrektur und Paritybit

Das Paritybit am jeweiligen Ende eines Zeichen gibt nur Auskunft darüber ob das gerade übertragene Zeichen falsch ist oder nicht. Nachrichten restaurieren lassen sich mit dieser Information alleine nicht. Mir ist kein Verfahren im Amateurfunk bekannt das Paritybits zur Informationssicherung benutzt.

3.3 Datenrestaurierung bei AMTOR

Wie im Punkt 2.1.3 schon beschrieben, kann man bei der Übertragung im AMTOR-Verfahren am Zeichen selbst erkennen ob es fehlerhaft ist oder nicht (4 mal 1, 3 mal 0). Das alleine reicht – wie beim Paritybit – zur Restaurierung nicht aus. Deshalb gibt es bei AMTOR zwei grundsätzlich unterschiedliche Verfahren zur Fehlerkorrektur:

- das FEC-Verfahren (Foreward Error Correction) für den 1:N – Betrieb und
- das ARQ-Verfahren für den 1:1 Betrieb.

Beim FEC-Verfahren werden die Zeichen doppelt übertragen. Um die Übertragung noch etwas besser abzusichern wird die zu sendende Nachricht kopiert, um 3 Zeichen gegeneinander verschoben und die Zeichen reissverschlussartig zusammengefasst.

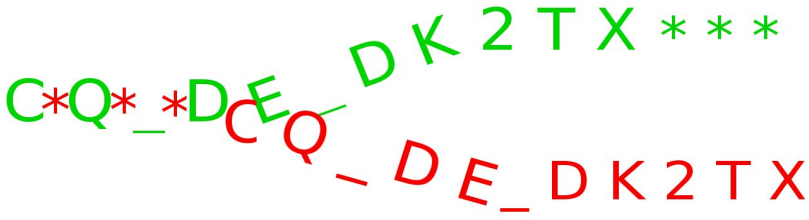

Bild 3: Verschachtelung des doppelt übertragenen Textes bei AMTOR. Die Sternchen „*“ sind Füllzeichen die nicht abgedruckt werden.

Damit ist die Wahrscheinlichkeit größer, dass eine zeitlich kurze Übertragungsstörung nicht beide Zeichen (erste Information und Kopie) zerstört. Die „Korrektur“ bei diesem Verfahren besteht daraus, dass im Fall dass beide Zeichen fehlerfrei übertragen wurde eines davon genommen wird.

Wenn ein Zeichen nicht den 4 zu 3 – Vorgaben entspricht wird jeweils das andere genommen. Werden beide Zeichen als fehlerhaft erkannt wird eine Fehlerinformation (Schmierzeichen) ausgegeben.

Beim ARQ-Verfahren wird der gesamte Text in kleine Buchstabenblöckchen zerlegt. Diese werden einzeln übertragen und vom Empfänger einzeln positiv oder negativ quittiert. Bei positiver Quittung wir das nächste Blöckchen gesendet, bei negativer das letzte wiederholt. Zusätzlich dazu enthält das Übertragungprotokoll noch eine Menge anderer Reaktionen (z.B. für der Wechsel von Sender und Empfänger, Ausfall einer der beiden Stationen...)

3.4 Datenrestaurierung beim CRC-Verfahren

Das CRC-Verfahren hat man bei den früheren Magnetbandaufzeichnungen, zur Prüfung der aufgespielten Daten benutzt. Das Verfahren lässt sich mit einer sehr einfachen Logik aus Exklusiv-Oder – Gattern aufbauen. Man konnte damit parallel zum laufenden Schreibvorgang mitlesen, am jeweiligen Ende eines Datenblocks erkennen ob die Daten korrekt aufgespielt waren und den Schreibvorgang im Fehlerfall wiederholen. Eine Auskunft, welche Zeichen in einem Datenblock fehlerhaft waren lässt sich mit dem CRC-Verfahren alleine nicht herausfinden. Beim Übertragungsverfahren OPERA wird das CRC in Zusammenspiel mit einem anderen Fehlererkennungsverfahren der Walsh-Hadamard Kodierung benutzt die Daten wiederherzustellen. Dazu wird die Nachricht (bei OPERA ist es nur das Call) in 3-Bit Stückchen zerhackt. Jedem dieser 8 möglichen 3-Bit-Stückchen wird dann eine 7-Bit lange Zahl zugeordnet.

Zu kodierende Nachricht	Kodierte Nachricht
000	0 0 0 0 0 0 0
001	1 0 1 0 1 0 1
010	0 1 1 0 0 1 1
011	1 1 0 0 1 1 0
100	0 0 0 1 1 1 1
101	1 0 1 1 0 1 0
110	0 1 1 1 1 0 0
111	1 1 0 1 0 0 1

Tabelle 2: Der bei OPERA verwendete Walsh-Hadamard-Code mit einem minimalen Hamming-Abstand von 4

Diese 7-Bit-Blöcke werden dann aneinandergehängt über das gesamte Bitmuster eine CRC-Summe berechnet. Haben sich während der Übertragung nun mehrere Fehler eingeschlichen, dann kann man schrittweise versuchen zuerst die als fehlerhaft erkannten 7-Bit-Blöcke gegen die mit größter Wahrscheinlichkeit korrekten Blöcke tauschen (von den möglichen 128 7-Bit-Blöcken sind nur 8 zulässig) bis alle 7-Bit – Blöcke korrekt sind. Von diesem korrigierten Datensatz kann man die Prüfsumme bestimmen. Unterscheidet sie sich trotzdem von der übertragenen Prüfsumme, lässt sich aus der Anzahl der Unterschiede zwischen beiden Prüfsummen mit einer gewissen Wahrscheinlichkeit erkennen ob die Prüfsumme (auch) fehlerhaft übertragen wurde. Letztendlich läuft das Verfahren auf „probieren“ hinaus: Es werden die Bits die mit der höchsten

Wahrscheinlichkeit fehlerhaft übertragen wurden korrigiert – dann wird getestet ob die Prüfsumme stimmt. Danach werden die Tests mit Korrekturen wiederholt, die in der Wahrscheinlichkeit als nächste drankommen u.s.w. Dieses Verfahren muss natürlich irgendwann abgebrochen werden weil

- die Wahrscheinlichkeit dass die Restaurierung mit steigender angenommener Fehleranzahl zu richtigen Ergebnissen führt schnell kleiner wird
- die erforderliche Anzahl von möglichen Korrekturen rapide ansteigt.

3.5 Datenrekonstruktion beim Solomon-Reed-Verfahren

Wie schon beschrieben, wird das Solomon-Reed Verfahren im Amateurfunk bei der Betriebsart JT65 verwendet. In einer Beschreibung über das JT65-Verfahren steht, dass die Rekonstruktion gut funktioniert solange die Fehleranzahl in der Übertragung gering bleibt. Dafür gebe es eine Bibliothek von Funktionen zur Fehlerkorrektur dessen Funktionsweise nur durch viel Mathematik erklärt werden kann. Bei größerer Anzahl von Fehlern scheint es kein geschlossenes Verfahren mehr zu geben. In diesem Fall wird das Verfahren der „Brutalen Gewalt“ angewendet. Bei JT65 wird das Verfahren „Aggressive Deep Search“ bezeichnet.

3.5.1 Datenrestaurierung nach dem Verfahren der brutalen Gewalt

Wenn das „normale“ Verfahren der Fehlerkorrektur nicht mehr funktioniert gibt es noch eines, das die Tatsache nutzt, dass die Anzahl der möglichen Nachrichten um viele Größenordnungen über der Anzahl der gültigen Nachrichten liegt. Die Idee dabei ist die, dass man vermutete Nachrichten mit der Solomon-Read Information versieht und das Ergebnis mit dem empfangenen Signal vergleicht. Ist eine Nachricht dabei, deren Kodierung – verglichen mit den anderen Nachrichten – signifikant näher an der empfangenen Information liegt, geht man davon aus, dass diese eine Nachricht die ist, die gesendet wurde. Dieses Verfahren hat allerdings zwei gravierende Nachteile:

- Anzahl der zu kodierenden Nachrichten ist deutlich zu groß. Um es vollständig richtig zu machen, müsste man alle denkbaren QSOs mit allen denkbaren Rufzeichen kodieren und mit dem Empfangsmuster vergleichen. Diese Datenmenge wäre so groß, dass die Lebensdauer eines PCs nicht ausreichen würde um alle Fälle durchzuspielen (Rechenzeit: Mehrere 100 Millionen Jahre pro dekodierter Nachricht).
- Eine auf diese Weise gefundene Nachricht kann auch völlig falsch sein. So kann z.B. weißes Rauschen für ein Bitmuster sorgen, dass ganz zufällig in der Nähe einer der kodierten Nachrichten liegt und damit als genau diese Nachricht interpretiert wird.

In der Betriebsart JT65 – die dieses Verfahren nutzt gibt es Möglichkeiten auch diesen zwei Nachteilen effektiv zu begegnen:

Um den ersten Nachteil in den Griff zu bekommen muss die Menge der Nachrichten deren Kodierung verglichen werden soll stark eingeschränkt werden. So wird z.B. beim EME-Betrieb von Standard-QSOs ausgegangen. Außerdem wird die Liste der Rufzeichen auf die beschränkt, die irgendwann schon mal über EME gehört wurden. Mit diesen Einschränkungen ist es dann möglich mit vertretbaren Rechenzeiten an die gewünschten Nachrichten zu kommen.

Das zweite Problem lässt sich nur aus der Welt schaffen indem man wartet, ob hintereinander

ähnliche Nachrichten dekodiert werden wie z.B. zwei CQ-Rufe mit demselben Rufzeichen. Nachdem die JT65-Meldungen nur entweder komplett richtig oder komplett falsch sein können, bedeutet die Tatsache dass zwei Nachrichten die das gleiche Rufzeichen enthalten richtig dekodiert worden sein müssen. Die Wahrscheinlichkeit zufällig zwei Nachrichten zu dekodieren die dasselbe Call betreffen ist so gut wie unmöglich.

Das Ganze vielleicht noch in Zahlen:

Eine JT65-Nachricht besteht aus 72 Bit. Damit können also $2^{72}=4{,}7*10^{21}$ mögliche Nachrichten gesendet werden.

Aus den 72 Bit werden 63 sechs Bit-Symbole gemacht. Damit können $64^{63}=6{,}2*10^{113}$ mögliche Bitmuster empfangen werden oder anders ausgedrückt:

Auf eine gültige Nachricht kommen $6{,}2*10^{113}/4{,}7*10^{21}=1{,}3*10^{92}$ fehlerbehaftete Bitmuster von denen ein gewisser Anteil einer gültigen Nachricht zugeordnet werden kann.

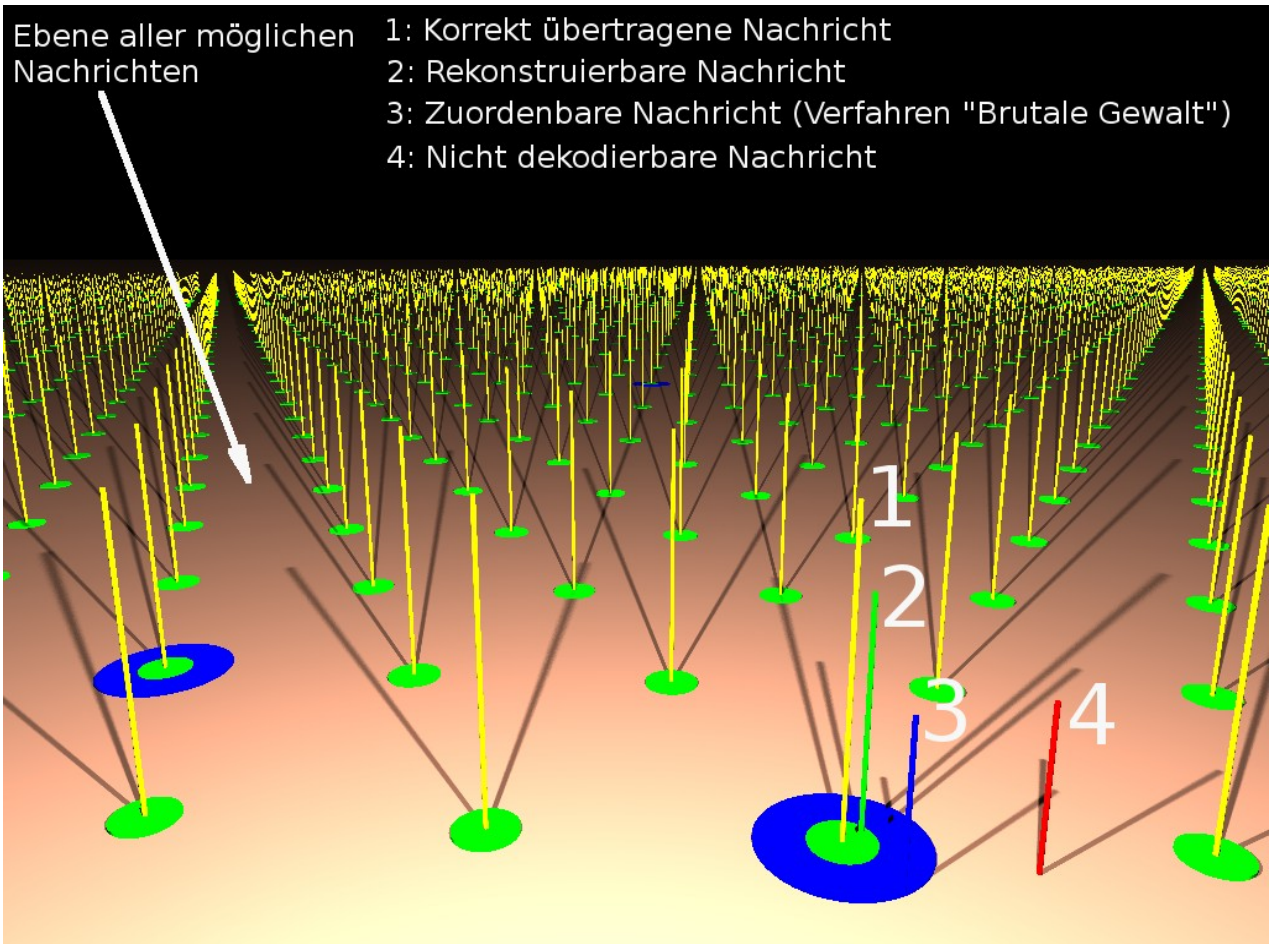

Bild 4: Darstellung der Möglichkeiten der Datenrekonstruktion beim Solomon-Reed-Verfahren:

Die gelben Nadeln sollen die Empfangsdaten (Bitmuster) darstellen die einer Nachricht bei fehlerfreiem Empfang entsprechen.

Die grünen Bereiche bezeichnen die Stellen von denen aus eine direkte Datenrekonstruktion möglich ist.

Die blauen Bereiche markieren die Flächen deren Empfangsdaten der gelben Nachricht im Zentrum zugeordnet werden können. Wegen der extrem großen Anzahl möglicher Nachrichten ($4{,}7*10^{21}$)

werden nur wenige ausgewählte Nachrichten für das Verfahren der „brutalen Gewalt“ verwendet.

Außerhalb dieser Bereiche ist keine Dekodierung möglich.

3.6 Datenrekonstruktion von Faltungscodes mit dem Viterbi-Dekoder

Wie bereits im Punkt 2.1.6 beschrieben hat es von der Beschreibung der Erzeugung von Faltungscodes bis zur Beschreibung einer optimalen Dekodierung 12 Jahre gedauert. Beim optimalen Dekodierungsverfahren – dem Viterbi-Dekoder wird versucht den Inhalt des zentralen Schieberegisters zu jedem Zeitpunkt zu rekonstruieren. Nachdem auf der Senderseite die Originalnachricht in das zentrale Schieberegister eingetaktet wird, kennt man die Originalnachricht wenn man den Inhalt des Schieberegisters zu allen Zeitpunkten kennt. Das Dekodierverfahren lässt sich am besten mit einer graphischen Methode darstellen. Dazu verwendet man ein Diagramm in dem die Zustandsänderungen des Schieberegisters dargestellt werden, das Trellis-Diagramm.

Das Trellis-Diagramm besteht aus einer Anzahl von Zeilen – eine Zeile pro möglichem Zustand des Schieberegisters und eine Spalte pro Takt. Der Zustand des Schieberegisters wird einfach ermittelt in dem man die Bitfolge der Registerzellen als binäre Zahl interpretiert. Diese Zahl entspricht dann der Zeilennummer im Diagramm. Schließlich gibt es noch die stillschweigende Vereinbarung, dass alle Zellen des Schieberegisters zu Beginn den Wert 0 haben. Damit beginnt das Diagramm mit einem Punkt in der Zeile 0 zum Zeitpunkt 0. Wenn wir uns die im Bild 2 dargestellte Struktur des QPSK31-Faltungsencoders ansehen, erkennen wir dass für jedes Bit der Originalnachricht zwei Bits übertragen werden. In dem Falle der QPSK31-Kodierung werden aus einer zu übertragenden „0“ oder „1“ je nach aktuellem Schieberegisterzustand die Bitmuster „00“ bis „11“ (auch Symbole 0 bis 3 genannt). In der folgenden Tabelle sind für alle möglichen Registerzustände und Nachrichten die zugehörigen Symbole und der resultierende Registerzustand gezeigt.

Alter Registerzustand	Nachricht	Übertragenes Symbol	Neuer Registerzustand
0	0	1	0
0	1	2	1
1	0	3	2
1	1	0	3
2	0	3	4
2	1	0	5
3	0	1	6
3	1	2	7
4	0	0	8
4	1	3	9
5	0	2	10
5	1	1	11
6	0	2	12
6	1	1	13

7	0	0	14
7	1	3	15
8	0	2	0
8	1	1	1
9	0	0	2
9	1	3	3
10	0	1	4
10	1	3	5
11	0	2	6
11	1	1	7
12	0	3	8
12	1	0	9
13	0	1	10
13	1	2	11
14	0	1	12
14	1	2	13
15	0	3	14
15	1	0	15

Tabelle 3: Kodierung von Nachrichten mit dem in Bild 2 dargestellten Faltungscode.

Für die Dekodierung geht man wie folgt vor:

Man sieht in der Tabelle nach, ob das empfangene Symbol bei dem augenblicklichen Registerinhalt möglich ist. Ist es möglich, kann man im Trellis-Diagramm eine Linie zum neuen Registerinhalt ziehen. Wird ein Symbol empfangen, das zum augenblicklichen Registerinhalt nicht passt, muss wenigstens ein Übertragungsfehler eingetreten sein. In dem Fall verzweigt man die Linie zu den beiden Zuständen die zur der Originalnachricht 0 oder 1 gehören. Außerdem versieht man die Linien an der Verzweigung mit einer Kennung, dass ein Übertragungsfehler stattgefunden hat. Beim nächsten empfangenen Symbol verfährt man genauso nur dass man jetzt nicht nur von einem Registerinhalt ausgeht sondern von allen bis dahin entstandenen Endpunkten im Diagramm.

Ist während der gesamten Übertragung nur ein Fehler aufgetreten, dann wird im Trellis-Diagramm ein Pfad (Linie) entstehen, der insgesamt einen Fehler anzeigt. An der Verzweigung die durch den Übertragungsfehler generiert worden ist beginnt ein weiterer Teilpfad dessen vermuteter Registerinhalt nicht mit dem tatsächlichen übereinstimmt. Die empfangenen Symbole passen deshalb im Mittel in der Hälfte der Fälle nicht zu den in diesem Teilpfad vermuteten Registerinhalten was zu weiteren Verzweigungen und zusätzlichen Fehlern führt. Am Ende der Übertragung (mit nur einem angenommen Übertragungsfehler) wird es deshalb neben dem einen Pfad mit einem erkannten Fehler eine große Zahl sich fein verästelnder Pfade geben deren Fehlersummme mit jedem Takt stetig angestiegen ist. Sucht man sich am Ende der Übertragung den Pfad mit der geringsten Fehlerzahl heraus, ist die Wahrscheinlichkeit sehr groß, das dieser Pfad die

korrekte Nachricht repräsentiert.

Es ist leicht nachzuvollziehen, dass die Anzahl der Pfade im Laufe einer Übertragung sehr schnell ansteigt was das Verfahren unhantierbar machen würde. Während der Übertragung werden deshalb die Pfade gelöscht, die eindeutig von einem nicht synchronen Registerinhalt ausgehen und deshalb stetig ansteigende Fehleranzahlen zeigen.

Um auch die letzten zu übertragenden Bits mit derselben Sicherungsinformation zu versehen wie die übrigen, wird an die Nutzdaten jeweils ein Nachspann gehängt der so viele Nullen enthält die das Schieberegister Bitstellen hat. Damit hat man für die letzten übertragenen Symbole noch ein zusätzliches Kriterium zur Fehlererkennung: Da die letzten Bis alle den Wert „0“ haben ist dort jeweils nur das Symbol zur „0“ zulässig. Man braucht also keine zusätzlichen Verzweigungen mehr einbauen sondern kann von der Nachricht „0“ ausgehen. Ist das empfangene Symbol korrekt, gehört aber zu einer „1“, dann müssen gleich zwei Übertragungsfehler aufgetreten sein, ist es nicht korrekt, bedeutet das einen Übertragungsfehler.

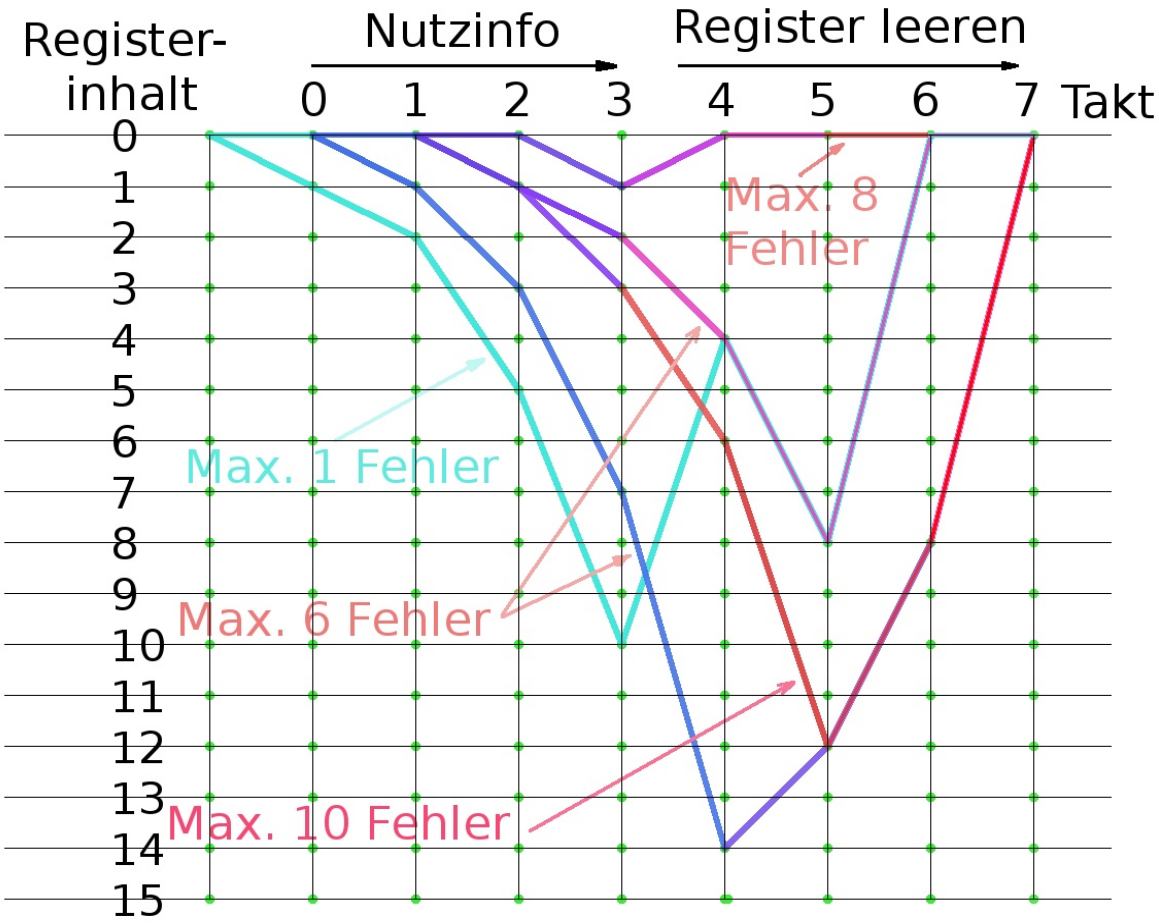

Bild 5: Trellis-Diagramm zum Faltungscode aus Bild 2 bei der Übertragung der Nutzdaten 10100000 und einem angenommenen Übertragungsfehler im ersten Symbol.

Dieses Verfahren funktioniert natürlich auch dann wenn während der Übertragung mehrere verteile Fehler aufgetreten sind.

3.6.1 Probleme bei der Dekodierung von Faltungscodes

Es ist nicht einfach zu entscheiden ab welcher Fehlersumme eine Nachricht als so unzuverlässig zu bezeichnen ist, dass die Dekodierung mit der Kennung „Nicht dekodierbar“ abgebrochen werden muss. Es kann deshalb durchaus vorkommen, dass aus dem Dekoder eine absolut falsche Nachricht kommt. Das Problem sieht man gelegentlich bei der Betriebsart WSPR. Dort kann es vorkommen dass „Spots“ auftauchen deren Rufzeichen nicht zum Lokator passen und deren Leistungsangaben völlig unrealistisch sind.

Aus der Beschreibung des Punktes 3.6 geht hervor, dass sich ein Pfad verzweigt, wenn bei zwei direkt aufeinander folgenden Bits ein Fehler erkannt wird. In dem oben gezeigten Beispiel wenn aus „01“ ein „00“ oder ein „11“ wird. Tauchen aber gleich zwei Fehler hintereinander auf, in unserem Fall aus der „01“ eine „10“ wird, dann wird im Trellis-Diagramm zu dem Zeitpunkt der falsche Zustand und gleichzeitig die Fehleranzahl 0 eingetragen. Dieser Fehler bleibt dann erhalten führt zu weiteren Folgefehlern – auch dann wenn keine weiteren Übertragungsfehler mehr aufgetreten sind. Natürlich lässt sich auch gegen dieses Problem eine Lösung finden (z.B. indem man auch beim scheinbar fehlerfreien Fall eine Verzweigung mit entsprechend hoher Fehleranzahl einfügt was den Rechenaufwand deutlich erhöht). Trotzdem ist es so dass Faltungscodes auf Bündelfehler (also mehreren Fehlern direkt hintereinander) empfindlich reagieren.

Faltungscodes haben den Vorteil, dass sich die Sicherung der Übertragung durch verlängern des Schieberegisters fast beliebig verbessern lässt. Der Zusatzaufwand bleibt dabei auf der Sendeseite gering, auf der Empfangsseite steigt er dafür exponentiell an. So hätte ein vollständiges Trellis-Diagramm für WSPR (32-Bit Schieberegister) $2^{32} = 4294967296$ unterschiedliche Zustände. Entsprechend groß kann auch die Anzahl der Pfade werden. Für solche Kodierungen gibt es inzwischen andere Dekodierungsverfahren die zwar weniger Rechenaufwand erfordern, deren Aussagekraft aber nicht mehr der eines vollständigen Viterbi-Dekoders entspricht.

4. Verschlüsselung

Das deutsche Wort Verschlüsselung mag in dem Zusammenhang etwas irreführend klingen. Bei der hier beschriebenen Form der Verschlüsselung (im englischen „scrambling“) geht es nicht darum die Nachricht gegen unberechtigtes lesen zu schützen. Mit dieser Art der Verschlüsselung sollen zwei Dinge erreicht werden:

- Eine eventuell mittelwertbehaftete Nachricht soll mittelwertfrei gemacht werden.
- Herausragende Frequenzanteile bei strukturierten Nachrichten sollen geglättet werden.
- In Nachrichten eventuell auftretende Folgen von Nullen sollen vermieden werden wenn in der weiteren Verarbeitung Codes benutzt werden bei denen der Hamming-Abstand zur Null kleiner ist als zu anderen Werten.

Signale die mittelwertfrei sind und eher weißem Rauschen ähneln sind besser zu übertragen als solche, die signifikant hervortretende Frequenzanteile haben.

Die Verschlüsselung geschieht dadurch dass die Nutzinformation durch eine Exlusiv-ODER-Funktion mit einem Pseudorauschsignal verknüpft wird. Entschlüsseln lässt sich das Ergebnis einfach dadurch dass man die verschlüsselte Nachricht mit derselben Rauschfolge per EXOR verknüpft.

Im Amateurfunk sind mir zwei Verfahren bekannt bei denen diese Art der Verschüsselung angewendet wird:

- Bei der Übertragung von Telemetriedaten von Amateurfunksatelliten
- Im Verfahren OPERA

Beispiel der Verschlüsselung der Folge 11111000001111100000... mit der OPERA Rauschfolge

1 1 1 1 1 0 0 0 0 1 1 1 1 1 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 1 1 1 1 1 0 0 0 Nachricht

0 1 1 1 0 0 0 0 1 0 1 0 1 0 1 1 1 1 1 1 0 0 1 1 0 1 1 0 1 0 0 0 0 0 0 0 1 1 0 0 1 0 0 0 1 0 1 0 Rauschfolge

1 0 0 0 1 0 0 0 1 0 0 1 0 1 0 1 1 1 1 1 1 1 0 0 1 1 1 0 1 0 1 1 1 1 1 0 1 1 0 0 0 1 1 1 0 0 1 0 Ergebnis

Wiederholt man diese Operation mit dem Verschlüsselungsergebnis und derselben Rauschfolge erhält man wieder die ursprüngliche Nachricht:

1 0 0 0 1 0 0 0 1 0 0 1 0 1 0 1 1 1 1 1 1 1 0 0 1 1 1 0 1 0 1 1 1 1 1 0 1 1 0 0 0 1 1 1 0 0 1 0 Ergebnis

0 1 1 1 0 0 0 0 1 0 1 0 1 0 1 1 1 1 1 1 0 0 1 1 0 1 1 0 1 0 0 0 0 0 0 0 1 1 0 0 1 0 0 0 1 0 1 0 Rauschfolge

1 1 1 1 1 0 0 0 0 1 1 1 1 1 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 1 1 1 1 1 0 0 0 Nachricht

Beim OPERA-Varfahren ist mir allerdings absolut rätselhaft warum dort überhaupt eine solche Verschlüsselung vorgenommen wird. Die in dem Verfahren nachgeschaltete Walsh-Hadamard-Codierung hat für alle Codes – also auch die Null den gleichen Hamming-Abstand. Damit ist eine vorgeschaltete Verschlüsselung eigentlich sinnlos.

5. Verwürfelung

Beim Funk tritt häufig das Problem auf, dass die Funkstrecke für gewisse Zeitintervalle ausfällt. Das kann durch QSB oder durch zeitlich begrenzte Störungen verursacht sein. Solche zeitlich begrenzten Ausfälle haben bei digitaler Datenübertragung oft die Wirkung dass mehrere direkt hintereinanderliegende Bits gemeinsam ausfallen. Diese Art von Fehlern – sogenannte Bündelfehler machen bei vielen Kodierungen Probleme bei der Datenrestaurierung. Ein Verfahren solchen Bündelfehlern zu begegnen wurde schon im Punkt 3.3 beschrieben. Bei AMTOR/FEC wird dieselbe Zeichenfolge zeitlich versetzt doppelt übertragen. Bei Verfahren, die zeitlich gleichmäßig verteilte Fehler gut korrigieren können, dafür aber bei Bündelfehler empfindlich reagieren (wie bei den Faltungscodes) gibt es die Möglichkeit die Bitreihenfolge so zu verändern, dass logisch hintereinanderliegende Bits zeitlich versetzt übertragen werden. Für diese Verwürfelung (engl. Interleaving) gibt es eine Reihe von verschiedenen Verfahren:

5.1 Verwürfelung mit Hilfe einer Matrix

In diesem Fall werden die Bits der zu verwürfelnden Information **zeilen**weise in eine Matrix geschrieben. Ist die Matrix gefüllt, werden die Bits **spalten**weise ausgelesen und so hintereinander aufgereiht.

Beispiel: Die verwendete Matrix soll aus 8x8 Bits bestehen:

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23

24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

Bild 5: Verwürfelung mit Hilfe einer 8x8 Matrix

Haben die zu verwürfelnden Daten vor der Operation die Folge 0, 1, 2, 3, 4, 5, 6, 7, 8, 9,.... 63, dann ist es nach der Operation die Folge 0, 8, 16, 24, 32, 40, 48, 56, 1, 9, 17, 25, 33, … 63.

Würden bei einer Übertragung die ersten 4 Bits gestört werden, wären das nach der Verwürfelung die Bits an den Stellen 0, 8, 16 und 24. Aus dem Bündelfehler wären damit vier einzelne isolierte Bitfehler geworden.

OPERA benutzt dieses Verfahren und verwendet dafür eine 7x17 Matrix.

5.2 Verwürfelung durch inverse Adressierung

Eine andere Methode ist die, die Bitadresse „von hinten“ zu lesen. Damit würde das Bit 01 nach der Verwürfelung an der Stelle 10 stehen, das Bit 14 an der Stelle 41 u.s.w

Adresse binär	Adresse dezimal	Inverse Adresse binär	Inverse Adresse dezimal
000	0	000	0
001	1	100	4
010	2	010	2
011	3	110	6
100	4	001	1
101	5	101	5
110	6	011	3
111	7	111	7

Bild 6: Beispiel einer inversen Adressierung für eine Adresslänge von 3

Aus einer Bitfolge mit den Stellen 0, 1, 2, 3, 4, 5, 6, 7 würde dann eine mit der neuen Reihenfolge 0, 4, 2, 6, 1, 5, 3, 7 entstehen. Ein Bündelfehler der die ersten drei Stellen betrifft würde sich damit nach der Verwürfelung auf die Stellen 0, 2 und 4 verteilen.

Ein solches Verfahren wird bei WSPR angewendet um den Viterbi-Dekoder gegen Bündelfehler zu schützen.

6. Der Autokorrelationsvektor

Die oben beschriebenen Verfahren funktionieren nur dann, wenn der Beginn einer Übertragung bitgenau getroffen wird und Sender und Empfänger während der gesamten Nachrichtenübertragung synchron bleiben. Um das zu verdeutlichen hier ein paar Beispiele:

6.1 Auswirkung einer um ein Bit verschobenen Übertragung bei der Verwürfelung

In dem Beispiel 5.1 wurden die Bits 0 bis 63 in mit Hilfe einer 8x8 Matrix verwürfelt. Verpasst der Empfänger das erste Bit, wird er in der Matrix links oben nicht das erste gesendete Bit „1“ sondern das erste empfangene Bit „8“ eintragen. Anstatt des ursprünglich gesendeten Strings 0,1,2,3,4,5,6... 62,63 würde bei der Restaurierung die Folge 8,9,10,11,12...63,1,2,3,4,5,6,7,x stehen wobei x das erste Bit aus dem nächsten 64-Bit Block wäre. Für einen Betrachter würden sich Teile der Information wiedererkennen lassen, eine ernsthafte Dekodierung (z.B. mit einem Viterbidekoder) wird absolut unmöglich. Bei der Verwürfelung nach der Methode der inversen Adressierung ergibt ein ähnliches Bild.

6.2 Auswirkung einer um ein Bit verschobenen Übertragung bei der Entschlüsselung

Untersucht man die Reaktion der in Punkt 4 beschriebenen Verschlüsselungsoperation auf eine solche Verschiebung um ein Bit kommt man zu folgendem Ergebnis:

0 0 0 1 0 0 0 1 0 0 1 0 1 0 1 1 1 1 1 1 1 0 0 1 1 1 0 1 0 1 1 1 1 1 0 1 1 0 0 0 1 1 1 0 0 1 0 x verschobenes Ergebnis

0 1 1 1 0 0 0 0 1 0 1 0 1 0 1 1 1 1 1 1 1 0 0 1 1 0 1 1 0 1 0 0 0 0 0 0 0 1 1 0 0 1 0 0 0 1 0 1 0 Rauschfolge

0 1 1 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 1 1 1 1 1 1 1 1 1 0 1 0 1 0 0 0 1 1 0 1 1 1 x „ist“ Ergebnis

1 1 1 1 1 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 1 1 1 1 1 0 0 0 0 „soll“ Ergebnis

Wie man hier sieht, hat das Ergebnis der „Entschlüsselung“ mit der Originalnachricht nichts mehr gemeinsam. Eine Weiterverarbeitung ist damit absolut unmöglich.

6.3 Synchronisierung mit Hilfe einer Pseudorauschfolge

Wie aus den oben gezeigten Beispielen zu erkennen ist, verlangen die beschriebenen Verfahren eine exakte Synchronisation zwischen Sender und Empfänger d.h. das Bit 1 der Sendung muss auch das Bit 1 beim Empfang sein, entsprechendes gilt natürlich auch alle anderen Bits. Laufen die Uhren auf Sende- und Empfangsseite mit genügender Genauigkeit, reicht es aus, den Start – also den Zeitpunkt des ersten Bits genau zu erfassen. Erschwerend kommt hinzu, dass das empfangene Signal oft selbst schon tief im Rauschen steckt und sich (wegen der Verschlüsselung) selbst wie weißes Rauschen anhört. Per Funkuhr zu bestimmten Zeitpunkten ausgelöste Starts reichen alleine nicht aus weil die nicht vorhersagbaren Reaktionszeiten der Betriebssysteme und die Signallaufzeiten die zeitliche Länge eines Bits meist übersteigen. Lösen lässt ich das Problem mit Hilfe einer Pseudorauschfolge – dem Autokorrelatationsverktor. Die Rauschfolge wird so auf die zu sendende Nachricht aufgeprägt, dass sie sich eindeutig von der eigentlichen Nachricht unterscheidet und leicht zu extrahieren ist.

Während des Empfangs schiebt man nun die aus der Nachricht extrahierte Rauschfolge Bit für Bit über den bekannten Autokorrelationsvektor und vergleicht bei jedem Takt alle Bit des

Autokorrelationsvektors mit dem empfangenen Bitstring. Dazu setzt man einen Zähler zu Beginn des Vergleichs auf Null, sind zwei übereinanderliegende Bits gleich, erhöht man den Zähler um 1, sind sie ungleich, zieht man von dem Zähler 1 ab. Die Zahl am Ende des Vergleichs ist ein Maß für die Übereinstimmung von eigenem Autokorrelationsvektor und empfangener Bitfolge.

```
0  0  0  0  1  0  0  1  1  0  0  0  1  0  1  1  0  1  0  0  0  Autokorrelationsvektor
0  1  0  0  0  1  0  1  0  0  1  0  1  0  1  0  0  1  0  1  1  Empfangene Rauschfolge
+  -  +  +  -  -  +  +  -  +  -  +  +  +  +  -  +  +  +  -  -
1  0  1  2  1  0  1  2  1  2  1  2  3  4  5  4  5  6  7  6  5  Zählerstand
```

Wie man aus dem Bild 7 erkennt, ist die Rauschfolge so gewählt, dass sie bei der Verschiebung „0“, also bei dem exakten Startzeitpunkt der Übertragung ein ganz signifikantes Maximum zeigt.

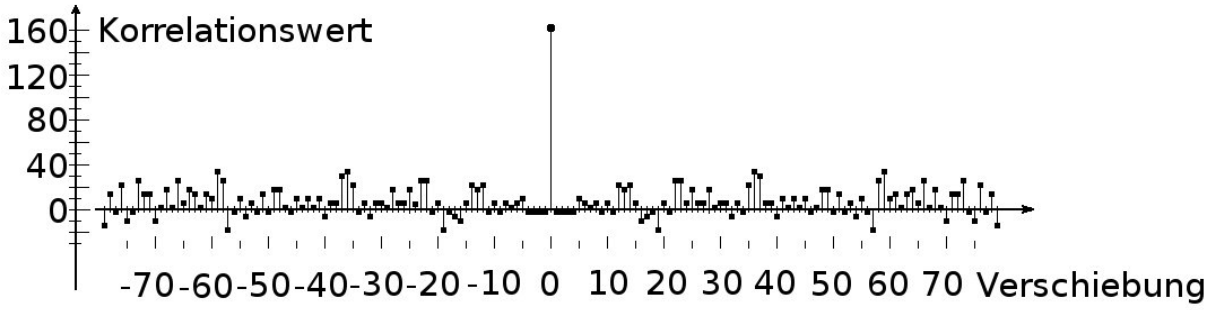

Bild 7: Korrelationswerte mit dem WSPR Autokorrelationsvektor

Bei WSPR ist dieser Autokorrelationsverktor so wichtig, dass er die Hälfte der übertragenen Information ausmacht: Bei WSPR werden vier unterschiedliche Töne im Wechsel ausgesendet. Ist das entsprechende Bit im Autokorrelationsvektor „0“, wird – abhängig von der Nutzinformation einer der unteren beiden Töne gesendet, ist das Bit „1“, einer der beiden oberen Töne.

7. Komprimierung

Mit den moderneren digitalen Betriebsarten hat auch die Datenkomprimierung im Amateurfunk Einzug gefunden. Dabei ist zwischen der Komprimierung von Text (Daten), Sprache (Audio) und Bildkomprimierung zu unterscheiden. Alle drei Sparten verwenden absolut unterschiedliche Verfahren. An der Stelle sollte noch angemerkt werden, dass sich komprimierte Daten meist nur dann wieder expandieren lassen, wenn die komprimierten Daten fehlerfrei übertragen wurden. Ein einzelnes falsch übertragenes Bit kann dazu führen dass ab der betroffenen Stelle keine Expandierung des Datensatzes mehr möglich ist. Bei der Datenübertragung per Funk muss also durch das jeweilige Übertragungsverfahren (z.B. Airmail, PACTOR oder Packet) sichergestellt werden dass die komprimierten Daten am Zielort fehlerfrei angekommen sind.

7.1 Komprimierung von Text (Daten)

Im Gegensatz zur Komprimierung von Bildern oder Sprache, muss die von Texten oder Daten „verlustfrei“ sein. Das expandierte Ergebnis muss eine 1:1-Kopie des originalen Datensatzes sein.

Im Allgemeinen komprimiert man Daten indem man in dem betroffenen Datensatz nach

Wiederholungen sucht und diese sich wiederholenden Teile nicht mehr 1:1 sondern nur Verweise darauf überträgt. Eine andere Möglichkeit ist, die zu übertragenden Zeichen entsprechend Ihrer Wahrscheinlichkeit (ähnlich wie bei CW) unterschiedlich lang zu machen und die Übertragung damit zu optimieren (Huffman Kodierung). Im Amateurfunk gibt es bei den zu übertragenden Texten zwei sehr unterschiedliche Klassen:

- Nachrichten mit genau festgelegter Zeichenanzahl und Zeichenumfang (WSPR, JT65, OPERA). Da es dort keine Wiederholungen gibt sind die sonst üblichen Verfahren hier nicht anwendbar.
- Nachrichten mit beliebig langem freiem Text. (PACTOR, Airmail)

7.1.2 Komprimierung von Text mit festem Umfang

Hat man nur eine kleine Zahl von Zeichen zu komprimieren bei denen möglicherweise auch noch der Zeichenvorrat festgelegt ist (z.B. nur die Zeichen A-Z oder nur die Ziffern 0-9) wie das bei WSPR, JT65 oder OPERA der Fall ist, dann ist dafür ein einfaches und sehr effektives Komprimierungsverfahren möglich:

- Man ordnet jedem möglichen Zeichen der ersten Stelle eine Zahl ab „0“ zu
- Man wiederholt diese Zuordnung mit allen weiteren Stellen und beginnt jeweils mit „0“
- Man multiplizert die Zahl die zum Zeichen der ersten Stelle gehört mit der maximal möglichen Anzahl der zweiten Stelle und addiert die Zahl die zum Zeichen der zweiten Stelle gehört dazu.
- Man multipliziert diese Summe mit der maximal möglichen Anzahl von Zeichen der nächsten Stelle und addiert die Zahl dazu die dem Zeichen dieser nächsten Stelle entspricht.
- Man setzt dieses Verfahren bis zur letzten Stelle fort.

Es ist leicht zu erkennen, dass das Ergebnis durch das fortgesetzte addieren und multiplizieren sehr schnell über alle hantierbaren Grenzen wächst. Diese Art der Komprimierung wird deshalb nur für Betriebsarten verwendet in denen nur ganz kurze Nachrichten wie Call, Lokator und Leistung (WSPR) oder nur das Rufzeichen (OPERA) oder maximal 13 Zeichen Text (JT65) gesendet werden. Bei längeren Nachrichten muss man – sofern Komprimierung überhaupt Sinn macht – auf andere Verfahren zurückgreifen.

Beispiel für eine stark vereinfachte Kodierung nach diesem Verfahren (ähnlich WSPR):

In dem Beispiel sollen nur Rufzeichen kodiert werden können, die folgende Struktur haben:

Stelle	Zeichenbereich	Zahlenbereich	Zahlenanzahl
1	A...Z	0...25	26
2	A...Z	0...25	26
3	0...9	0...9	10
4	A...Z	0...25	26
5	A...Z	0...25	26

Entsprechend der oben gezeigten Tabelle werden alle Buchstaben an allen Stellen in der gleichen Weise kodiert: A Entspricht der Zahl 0, B entspricht der Zahl 1 … Z entspricht der Zahl 25. Bei den Ziffern entspricht das Zeichen 0 der Zahl 0, das Zeichen 1 der Zahl 1 u.s.w.

Will man damit das z.B. das Rufzeichen DA1BC kodieren sieht die Berchnung so aus:

```
       D          A          1          B          C
(((((((3 * 26) + 0) * 10) + 1) * 26) + 1) * 26) + 2 = 527984
```

Die größte Zahl, die in dem Beispiel auftreten kann ist

```
       Z          Z          9          Z          Z
(((((((25 * 26) + 25) * 10) + 9) * 26) + 25) * 26) + 25 = 4569759
```

und lässt sich damit in 23 Bit darstellen. Als ASCII-Zeichen übertragen wären dafür statt dessen 5 * 8 = 40 Bit nötig.

7.1.3 Komprimierung von beliebigen Textdaten

Sieht man sich die Übertragung von Texten an (z.B. bei RTTY oder den ASCII-Code) stellt man fest, dass alle Zeichen die gleiche Länge haben: Bei RTTY 5 Bit, bei ASCII 7 oder 8 Bits. An dieser Stelle setzt die Komprimierung an. Ähnlich wie bei CW braucht man dafür Zeichen unterschiedlicher Länge. Je häufiger ein Zeichen ist, umso kürzer muss dessen Kodierung sein. Bei Zeichen unterschiedlicher Länge handelt man sich jetzt aber das Problem ein dass man irgendwie erkennen muss wo ein Zeichen aufhört und wo das nächste beginnt. Bei CW geschieht diese Erkennung durch den Abstand zwischen den Zeichen. So etwas ähnliches könnte man auch bei den Komprimierungsverfahren benutzen – also eine Bitfolge, die innerhalb eines Zeichens nicht auftaucht (z.B. 5 Nullen hintereinander). Dadurch verschenkt man aber wieder eine Menge Platz und die Komprimierung wird sehr uneffektiv. Die andere Lösung ist die, die Endekennung im Zeichen selbst zu verstecken. Am besten lässt sich das zugehörige Verfahren durch einen binären Entscheidungsbaum darstellen. Der Baum hat an jedem Knoten (also jeder Verzweigung) zwei Zweige. Der linke Zweig steht für eine „0“, der rechte für eine „1“. An den Enden der äußeren Zweige stehen zu kodierenden Buchstaben. Möchte man nun einen Buchstaben kodieren, geht man von der Wurzel aus die Zweige hoch bis zu dem gewünschten Buchstaben. Die Folge der dabei auszuführenden links- und rechts-Bewegungen ist die für den Buchstaben zu setzende Bitfolge. An diese Bitfolge kann die für den nächsten Buchstaben lückenlos angehängt werden. Bei der Dekodierung geht man genau umgekehrt vor: Man geht im Baum von der Wurzel aus entsprechend der Nullen und Einsen bis das erste „Blatt“ erreicht ist, übernimmt den Buchstaben und beginnt mit dem nächsten Bit wieder bei der Wurzel. Der Baum kann so aufgebaut werden, dass die Wege zu den häufigen Buchstaben kurz sind und umso länger werden je seltener das entsprechende Zeichen benutzt wird. Diese Form der Kodierung wird als Huffman-Kodierung bezeichnet und stammt aus dem Jahr 1952. Sie ist bei einer bekannten Häufigkeitsverteilung von Zeichen die effektivste Methode der Kodierung.

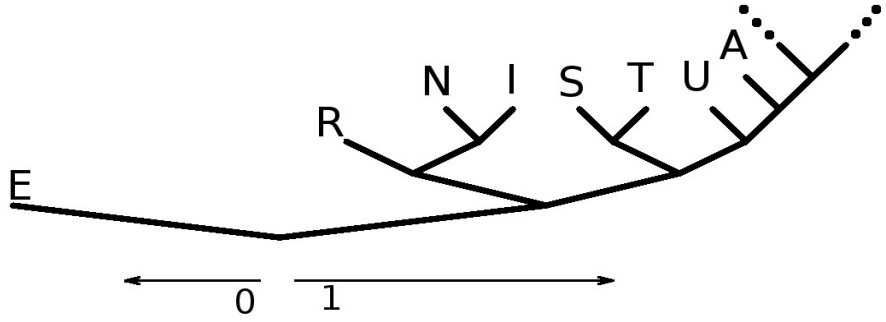

Bild 8: Teil eines unsymmetrischen Entscheidungsbaumes zur Komprimierung von deutschen Texten.

Nach Bild 8 würde die Kodierung des Namens ERNST das Bitmuster 0100101011001101 ergeben.

Hier zeigt sich allerdings schon das Problem der Expandierung bei Übertragungsfehlern: Wenn ein Bit falsch übertragen wurde, findet man im Baum nicht nur zu einem falschen Zeichen sondern wird im Allgemeinen auch zu einer anderen Zeichenlänge kommen. Damit sind von dem Fehler auch Folgezeichen betroffen. Würde in dem oben gezeigten Beispiel das zweite Bit fälschlicherweise als „0“ übertragen worden sein, dann würde bei der Dekodierung statt ERNST das Wort EEEENST entstehen wobei sich die Dekodierung (wie auch hier in dem Beispiel) nach einer Reihe von Zeichen wieder synchronisiert.

Ein weiteres Problem sind die Zeichenhäufigkeiten. Wenn die Art des Textes bekannt ist (wie z.B. deutscher Klartext) dann sind die Buchstabenhäufigkeiten bekannt. Sie beginnen mit E R N I S T U... Ist es kein deutscher Klartext ist der Versuch die Daten mit einem auf diesen Klartext optimierten Baum zu komprimieren sinnlos. Im ungünstigsten Fall braucht das Ergebnis sogar mehr Platz als die nicht „komprimierten“ Originaldaten.

Dem Problem lässt sich dadurch begegnen, dass der Baum „dynamisch“ wird. Dabei wird zuerst ein Baum z.B. mit lauter gleich langen Zeichen definiert. Während der Komprimierung wird nun mitgezählt wie häufig welches Zeichen ist. Zeigt sich nun ein Zeichen im Verhältnis zu den anderen Zeichen als besonders häufig, so wird der Baum ab einer bestimmten Schwelle so umgebaut, dass für das häufige Zeichen ein kürzerer Weg konstruiert wird und dafür das Zeichen mit der geringsten Häufigkeit einen längeren Weg bekommt. Auf diese Weise optimiert sich der Entscheidungsbaum im Laufe eines längeren Datensatzes selbst. Bei der Expandierung muss man natürlich dann genauso „mitzählen“ um den Baum zu denselben Zeitpunkten umbauen zu können wie es bei der Komprimierung gemacht wurde. Bei diesem Verfahren ist es noch weit wichtiger keine Übertragungsfehler zu haben denn jetzt kann schon ein einzelner Bitfehler dazu führen, dass die Entscheidungsbäume bei Sender und Empfänger unterschiedlich aussehen - eine Rekonstruktion der korrekten Daten wird damit fast unmöglich.

Leider handelt man sich bei dem oben beschriebenen dynamischen Baum gleich das nächste Problem ein: Was ist, wenn sich die Art des Textes innerhalb des Datensatzes ändert? Mit jedem neuen Zeichen ändert sich das Verhältnis Einzelzeichen zu Gesamtzeichen langsamer. Damit reagiert auch der Baumumbau immer langsamer auf Änderungen in den Verhältnissen der Zeichenanzahlen untereinander.

Die Abhilfe:Neuanfang. Nach einer bestimmten Anzahl von Zeichen wird wieder mit dem anfänglichen Standardbaum begonnen und die Zahlenverhältnisse zurückgesetzt. Damit werden die ersten Zeichen zwar noch recht uneffektiv komprimiert, der Baum passt sich aber recht schnell an die Zeichenverhältnisse des aktuellen Datenblocks an.

Weitere Komprimierung lässt sich erreichen wenn man nicht nur einzelne Zeichen sondern ganze Zeichengruppen zu einem Zeichen zusammenfasst oder indem man (wie z.B. bei dem Verfahren Airmail) für Leerzeichen eine Sonderbehandlung einführt. In diesem Verfahren wird davon ausgegangen, dass Leerzeichen oft nicht einzeln sondern in ganzen Gruppen auftreten. Bei Airmail werden direkt hintereinanderliegende Leerzeichen gezählt und dann nur ein Leerzeichen zusammen mit der Anzahl übertragen.

7.2 Komprimierung von Bilddaten

Für die Komprimierung von Bildern gibt es eine ganze Reihe verschiedener Verfahren. Einige davon verändern die eigentliche Bildinformation nicht. Mit diesen Verfahren lässt sich das Originalbild wieder rekonstruieren. Eine große Zahl anderer Verfahren verändert die Bildqualität. Das mit solchen Verfahren komprimierte Bild lässt sich dann nur noch mit geringerer Bildqualität restaurieren. Im Amateurfunk wird in dem DATV Verfahren u.a. MPEG-2 verwendet. Im MPEG-2 wird die Komprimierung von Bildern ähnlich gemacht wie bei dem weit verbreiteten JPEG-Format. Deshalb soll hier stellvertretend für andere Verfahren die JPEG-Komprimierung gezeigt werden.

7.2.1 Der Bildaufbau nicht komprimierter Bilder

In nicht komprimierter Form kann man Bilder als Matrix von Einzelpixeln darstellen wobei jedes Pixel (Bildpunkt) aus einer Mischung von drei Farben besteht. Stellt man jede dieser Teilfarben als Zahl zwischen 0 und 255 dar, kann man Bilder mit $256*256*256=16777126$ unterschiedlichen Farben erzeugen. Jeder Bildpunkt „verbraucht“ damit 3 Byte an Information. Ein digitales Photo mit 15 Millionen Bildpunkten braucht damit einen Speicherbereich von 45 Millionen Byte. In der Rechnerwelt ist die hier beschriebene Form der Darstellung als Bitmap „BMP“ bekannt.

7.2.2 Komprimierungsschritt 1: Umrechnung des Farbraums

Dieser „Trick“ mit der Umrechnung des Farbraumes wurde schon beim analogen Farbfernsehen angewendet. Dort hat man gelernt, dass man die Farbinformation, die man zu dem originalen schwarz/weiß-Bild hinzufügen muss eine viel kleinere Bandbreite haben darf als die für die Helligkeitsinformation. Bei der Komprimierung im JPEG-Verfahren wird das RGB (Rot, Grün, Blau)-Bild in ein YCbCr-Bild (Luminanz, Rot, Blau) umgerechnet. Die Werte Y, Cr und Cb sind einfache, aus den Größen R, G und B erstellte Linearkombinationen, die sich ebenso leicht wieder zurückrechnen lassen. Die einfachste Form der Darstellung einer solchen Art von Linearkombinationen ist die der Matrixform:

$$\begin{pmatrix} Y \\ C_b \\ C_r \end{pmatrix} = \begin{pmatrix} 0 \\ 128 \\ 128 \end{pmatrix} + \begin{pmatrix} 0{,}299 & 0{,}587 & 0{,}114 \\ -0{,}168736 & -0{,}331264 & 0{,}5 \\ 0{,}5 & -0{,}418688 & -0{,}081312 \end{pmatrix} * \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

Im folgenden Bild ist ein Originalbild in Bilder mit den drei Einzelkomponenten Y, Cr und Cb aufgespalten. Während das Y-Bild sehr detailreich aussieht, sind in den Bildern Cr und Cb nur wenig Kontraste zu erkennen. Halbiert man die Cb und Cr-Bilder jeweils in Höhe und Breite indem man in der Horizontalen jeden zweiten Bildpunkt weglässt und in der Vertikalen jede zweite Zeile, und rekonstruiert das Originalbild mit Hilfe dieser verkleinerten Farbinformationen ist so gut wie kein Qualitätsverlust zu bemerken. Das Gesamtbild braucht nach dieser Operation nur noch die Hälfte des ursprünglichen Platzes.

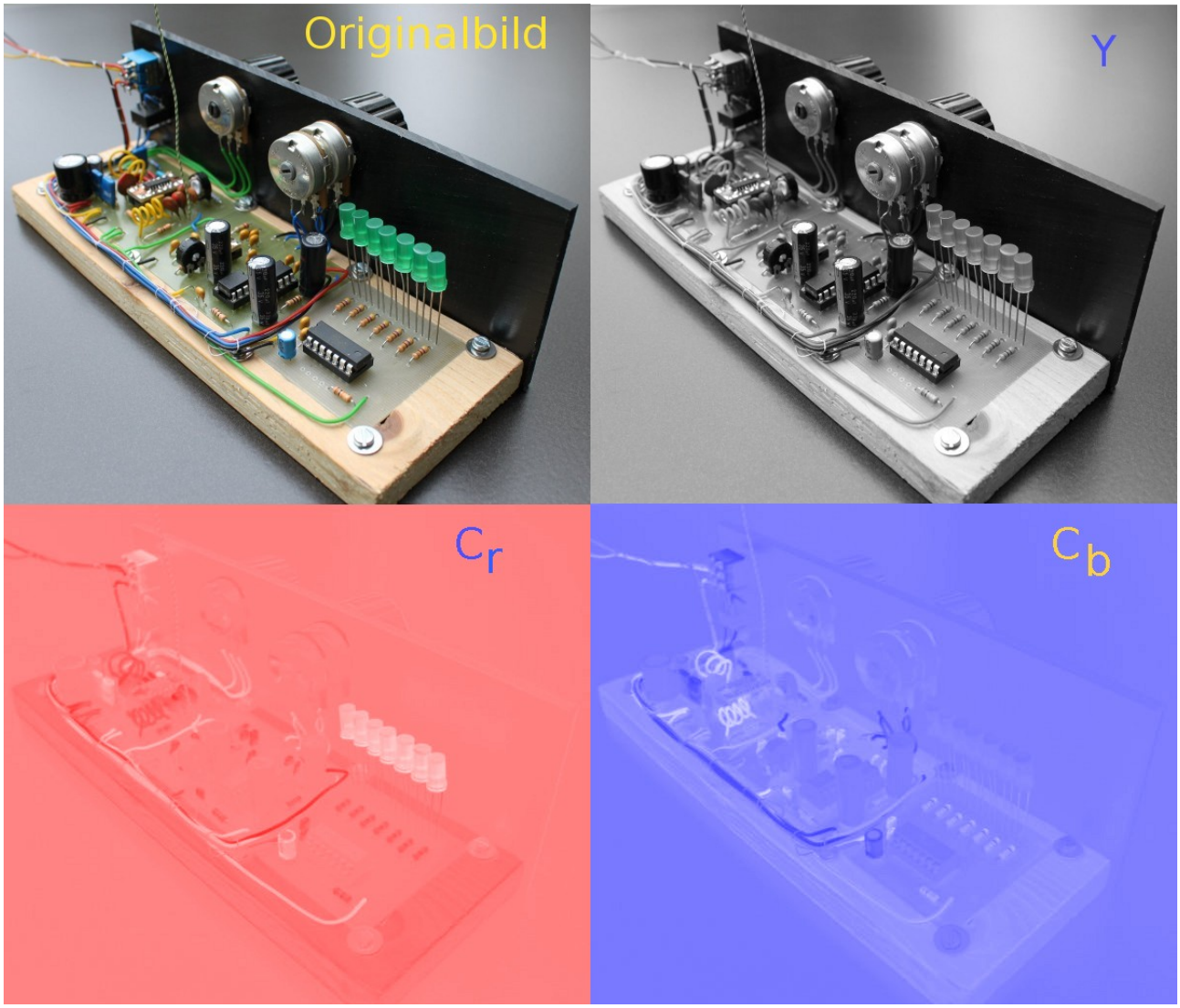

Bild 9: Aufteilung eines Bildes in die Y, Cb und Cr Komponenten.

7.2.3 Komprimierungsschritt 2: Die Kosinustransformation

Die Kosinustransformation ist eine Art der Fouriertransformation bei der nur reellwertige Zahlen auftreten. Die diskrete Kosinustransformation lässt sich deshalb mit dem für die FFT (Fast Fourier Transformation) bekannten Algorithmus realisieren. Eine Eigenschaft dieser diskreten Kosinustransformation (DCT) ist die, dass sie – auf die transformierten Daten angewendet – wieder zu den Originaldaten führt. Für die Komprimierung wird das Bild in Blöckchen von 8x8 Bildpunkten aufgeteilt. Für jeden Block werden nun die 64 Helligkeits- oder Farbinformation in 64

Koeffizienten umgerechnet. Sieht man jedes der Blöckchen als kleine 8x8 Matrix zeigt sich, dass die Koeffizienten links oben (also zu kleineren Frequenzen hin) deutlich größer sind als die rechts unten oder dass die langsameren Wechsel in Helligkeit oder Farbe weit stärker sind als die kleinflächigen (schnelleren). Der Vollständigkeit halber soll die bei der JPEG-Komprimierung verwendete DCT hier gezeigt werden:

$$F_{xy} = \frac{1}{4} C_x C_y \sum_{m=0}^{7} \sum_{n=0}^{7} f_{mn} \cos\frac{(2m+1)x\pi}{16} \cos\frac{(2n+1)y\pi}{16}$$

$$Mit\ C_x, Cy = \frac{1}{\sqrt{2}}\ wenn\ x, y = 0, sonst\ 1$$

$$F_{xy}: Koeffizienten\ und$$

$$fmn: Y, C_b\ oder\ C_r - Werte$$

Beispiel:

Als Beispiel wurde hier ein 8x8 Array von Bildpunkten aus der Mitte des oben gezeigten Y-Bildes genommen:

$$f = \begin{pmatrix} 158 & 160 & 159 & 170 & 201 & 218 & 225 & 225 \\ 161 & 161 & 161 & 169 & 201 & 219 & 226 & 227 \\ 163 & 161 & 158 & 165 & 195 & 217 & 226 & 227 \\ 163 & 161 & 158 & 162 & 192 & 214 & 224 & 228 \\ 165 & 163 & 158 & 163 & 191 & 212 & 223 & 228 \\ 166 & 164 & 161 & 163 & 191 & 210 & 222 & 228 \\ 162 & 163 & 159 & 163 & 189 & 209 & 220 & 227 \\ 154 & 156 & 154 & 160 & 185 & 207 & 220 & 227 \end{pmatrix}$$

Wendet man darauf die oben gezeigte Formel für die diskrete Kosinustransformation an kommt man zu folgendem Ergebnis:

$$F = \begin{pmatrix} 1502{,}1 & -215{,}1 & 47{,}6 & 38{,}7 & -9{,}9 & -14{,}4 & 0{,}2 & 6{,}1 \\ 14.6 & -4{,}8 & -11{,}9 & 7{,}6 & 0{,}3 & 0{,}0 & -0{,}7 & 1{,}0 \\ -4{,}1 & -4{,}3 & -8{,}1 & -2{,}5 & 0{,}3 & -0{,}2 & -1{,}1 & -0{,}6 \\ 5{,}6 & 5{,}0 & -1{,}3 & -0{,}6 & 1{,}1 & -0{,}9 & -0{,}5 & 1{,}0 \\ -6{,}4 & -3{,}6 & 0{,}1 & -0{,}4 & 0{,}1 & 0{,}2 & 0{,}7 & -1{,}1 \\ -0{,}6 & 1{,}3 & 0{,}6 & 0{,}2 & -0{,}3 & -0{,}4 & -0{,}2 & -0{,}5 \\ -1{,}7 & -0{,}7 & 0{,}9 & 0{,}2 & -0{,}3 & 0{,}9 & -0{,}1 & 0{,}2 \\ -0{,}1 & 0{,}6 & 0{,}1 & 0{,}5 & -0{,}1 & 0{,}4 & -0{,}6 & -1{,}1 \end{pmatrix}$$

Wie man hier gut erkennen kann, nehmen die Zahlen in ihrer Tendenz von links nach rechts und von oben nach unten betragsmäßig ab.

7.2.4 Komprimierungsschritt 3: Die Quantisierung

Die Matrix von Koeffizienten wird jetzt durch eine Matrix von Quantisierungswerten dividiert. Dabei steigen die Quantisierungswerte vom Gleichanteil Q_{00} bis zu dem Quantisierungswert für die höchste Frequenz Q_{77} stetig an. Je stärker die Komprimierung sein soll, desto größer sind die Quantisierungswerte.

Nach der Division werden die nun kleineren Koeffizienten auf die betragsmäßig nächst kleinere ganz Zahl gerundet. Nachdem die Koeffizienten zu den höheren Frequenzen hin sowieso schon kleiner sind als die zu den niedrigeren Frequenzen und diese kleineren Werte durch die größeren Quantisierungswerte dividiert werden, ist die Wahrscheinlichkeit sehr hoch, dass die Divisionsergebnisse dort kleiner „1“ werden, was nach der Abrundung den Wert 0 ergibt.

Beispiel einer Quantisierungsmatrix

$$Q=\begin{pmatrix} 5 & 8 & 12 & 16 & 26 & 33 & 41 & 50 \\ 8 & 10 & 14 & 20 & 27 & 34 & 42 & 51 \\ 12 & 14 & 18 & 23 & 29 & 36 & 44 & 53 \\ 16 & 20 & 23 & 27 & 33 & 40 & 47 & 56 \\ 26 & 27 & 29 & 33 & 38 & 45 & 52 & 60 \\ 33 & 34 & 36 & 40 & 45 & 51 & 57 & 65 \\ 41 & 42 & 44 & 47 & 52 & 57 & 64 & 71 \\ 50 & 51 & 53 & 56 & 60 & 65 & 71 & 78 \end{pmatrix}$$

Dividiert man nun F durch Q und rundet betragsmäßig auf ganze Zahlen ab, bleibt als Ergebnis:

$$F_Q=\begin{pmatrix} 300 & -26 & 3 & 2 & 0 & 0 & 0 & 0 \\ 8 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Nachdem der Wert links oben den „Gleichanteil“ - also die gemittelte Helligkeit des 8x8 Blockes angibt wird von diesem Wert der Gleichanteil des Blockes links davon abgezogen. Auf diese Weise werden allzu große Sprünge in der Helligkeit der Blöckchen vermieden.

7.2.5 Komprimierungsschritt 4: Die Sortierung

Sortiert man diese quantisierten Koeffizienten „meanderförmig“ $F_{Q00}, F_{Q01}, F_{Q10}, F_{Q20}, F_{Q11}, F_{Q02}, F_{Q03...}$ dann erhält man in dem Beispiel die Zahlenfolge 300,8,-26,3,1,0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0... 0. In dieser Beispielfolge sind ab dem 11. Koeffizienten

alle Koeffizienten (also 52 in Folge) dieses Blockes 0. Nachdem in einem „normalen“ Bild fast alle Blöcke so ähnlich aussehen lässt sich gut nachvollziehen dass sich solche Daten sehr effektiv mit der in Punkt 7.1.3 beschriebenen Komprimierungsmethode „eindampfen“ lassen.

7.3 Komprimierung von Audiodaten

Die Komprimierung von Audiodaten ist ein extrem komplexes Thema an dem schon über 45 Jahre geforscht und entwickelt wird. Entsprechend schwierig ist es nach so langer Zeit zu versuchen, auf den Zug mit aufzuspringen und selbst Software für die Komprimierung von Audioinformation zu schreiben. Nur so ist es wohl zu erklären wie der unsägliche Unfall passiert ist, dass Geräte mit einer Audiokompression auf den Markt gekommen sind (und noch verwendet werden) die den Chip einer Firma (AMBE) zur Komprimierung verwenden bei dem das verwendete Verfahren nicht offengelegt wird und das außerdem durch eine Reihe von Patenten geschützt ist. An dieser Stelle hat sich der Amateurfunk vom Funkdienst zu einer simplen Funkanwendung degradiert.

Glücklicherweise gibt es inzwischen Funkamateure die das Thema Audiokomprimierung selbst in die Hand nehmen und selbst Software für digitale Sprachübertragung schreiben. (z.B. FreeDV)

7.3.1 Klassifizierung der verschiedenen Methoden

Für die verschiedenen Anwendungen sind verschiedene Randbedingungen zu beachten was zu einer großen Anzahl absolut unterschiedlicher Verfahren führt. Folgende Randbedingungen (Anforderungen) kann man sich vorstellen:

- Die Komprimierung soll verlustfrei oder verlustbehaftet sein
- Komprimierung von Musik und, oder Sprache die nach der Expandierung genau so klingen soll wie das Original.
- Es soll nur Sprache mit möglichst guter Verständlichkeit übertragen werden
- Bei der zu kodierenden Sprache ist mit Hintergrundgeräuschen zu rechnen (oder nicht).
- Die Übertragung der komprimierten Daten erfolgt fehlerfrei.
- Es ist eine maximale Datenrate vorgegeben
- Es sind maximale Verarbeitungszeiten für Kodierung und Dekodierung vorgeschrieben.
- Die übertragenen Daten können Einzelfehler (z.B. durch Rauschen verursacht) haben.
- Die übertragenen Daten können Bündelfehler (verursacht durch QSB oder QRM) haben.

Ein frei zugängliches Verfahren „von der Stange“ wie z.B. das bei VoIP (voice over IP) verwendete kann man nicht nehmen weil die Randbedingungen, die der Übertragungskanal bei Funk bietet überhaupt nicht zu der Übertragung im Internet passen.

7.3.2 Die Realisierung

Bevor für den Amateurfunk geeignete Verfahren etwas näher beschrieben werden sollen hier kurz auch die anderen Verfahren gestreift werden:

Verlustfreie Kodierung

Für diese Form der Kodierung wird das schon im Punkt 7.1.3 beschriebene Verfahren der Huffman-Kodierung benutzt wobei man dabei besonders auf die Eigenheiten des Audio-Datenstroms eingeht. Damit kann man Kompressionsraten von 25% bis 70% erreichen.

Kompression von Sprache und Musik

Bei dieser Art den Kompression nutzt man aus, dass es eine Reihe von akustischen „Ereignissen“ gibt, die wir nicht wahrnehmen können wie z.B. leise Töne während oder kurz nach lauten Ereignissen oder die Phasenlagen der Einzelkomponenten im Frequenzbereich untereinander. Eine Methode der Kompression ist ähnlich der in Punkt 7.2. beschriebenen Bildkompression: Mit einer modifizierten Kosinustransformation setzt man Audioblöckchen vom Zeitbereich in den Frequenzbereich um, quantifiziert die so erhalten Koeffizienten und komprimiert das Ergebnis mit einer passenden Huffman-Kodierung.

Sprachkompression mit hoher Kompressionsrate

Hier geht es nicht mehr um eine Kompression die eine möglichst naturgetreu erscheinende Übertragung von Geräuschen ermöglicht sondern nur um die Übertragung von Sprache mit möglichst guter Verständlichkeit und parallel dazu möglichst geringer Übertragungsbandbreite.

Das extremste Komprimierungsverfahren wäre, die auf der Senderseite gesprochene Information in normalen geschriebenen Text umzuwandeln, als Text zu übertragen und sich auf den Empfängerseite per Rechner vorlesen zu lassen. Damit wäre natürlich jede Eigenheit des Sprechers verloren. Außerdem würde die Umwandlung eines gesprochenen Textes in Schriftform eine extreme Rechenleistung erfordern. Für das „Vorlesen“ gibt es schon seit langer Zeit verhältnismäßig einfache Software.

Für das Verfahren des „Vorlesens“ gibt es ein Modell das auch bei der hier verwendeten Sprachkompression eingesetzt wird. Man bildet den den menschlichen „Sprechapparat“ nach:

Letztendlich besteht dieser Apparat aus aus zwei verschiedenen Anregungen

- ein Tongenerator für die Vokale und
- ein Rauschgenerator für die Konsonanten
- Ein Filter der die akustischen Filtereigenschaften des Rachen- und Mundraums nachbildet.

Erstellt man nun einen Datensatz, der folgende Informationen enthält:

- Vokal oder Konsonant
- Lautstärke der Anregung (also des Rauschgenerators oder des Tongenerators)
- Filterparameter

und wiederholt die Ermittlung des Datensatzes in kurzen Zeitabständen (z.B. alle 20ms) dann lässt sich damit eine gut verständliche Sprache beschreiben. Die Kunst bei diesem Verfahren ist

- ein geeignetes Filter zu finden das mit möglichst wenigen Parametern eine möglichst natürliche Sprachwiedergabe ermöglicht
- ein Verfahren zu finden, mit dessen Hilfe sich aus der Sprache die Werte für die Datensätze auf der Sendeseite so extrahieren lassen dass sie zu einer möglichst gut verständlichen

Sprachwiedergabe führen.

Im Amateurfunk kommt dann noch der Punkt dazu, dass das Verfahren auch dann noch funktionieren muss, wenn in der Übertragung Bündelfehler auftreten. Schließlich gibt es noch das Problem dazu, dass bedingt durch die schon Jahrzehnte laufende Forschung und Entwicklung auf diesem Gebiet eine unüberschaubare Zahl von Patenten existiert die alle in irgendeiner Form „umgangen“ werden müssen.

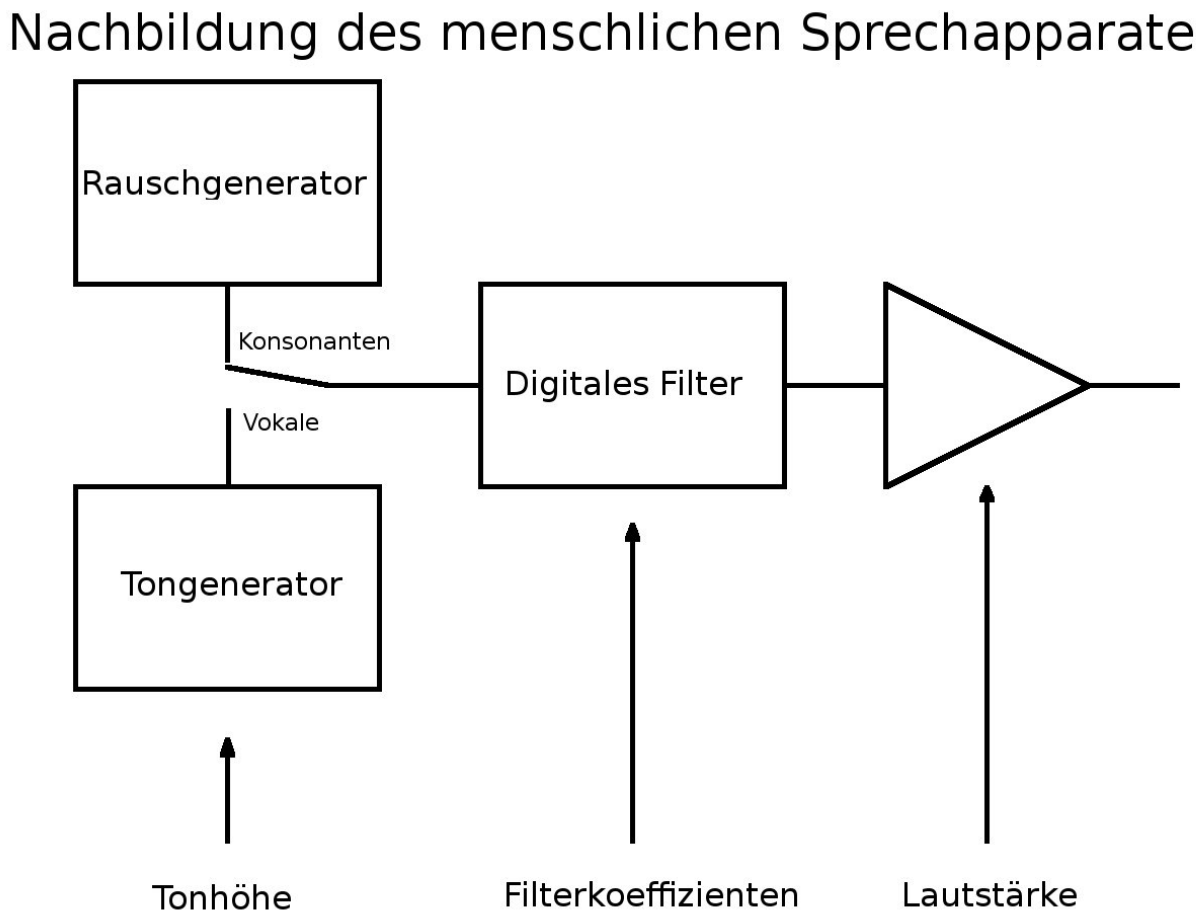

Bild 10: Aufbau des menschlichen Sprechapparates