Lineare Fehlerkorrigierende Codes

Ich werde zunächst einen kurzen Einblick in die Möglichkeiten der Fehlerkorrektur geben.
Danach befasse ich mich kurz mit Blockcodes bevor ich zu den Linearen Codes weitergehen werde.

Wir leben im Informationszeitalter und jeden Tag werden riesige Datenmengen von einem Punkt der Erde zum anderen durch Fernsehen, Internet, Handy, usw gesendet. Vor allem bei sensiblen Daten muss man sich dann natürlich darauf verlassen können, dass diese Daten korrekt sind. Der Bedarf an Systemen zur möglichst fehlerfreien Übertragung von Daten stieg daher in den letzten Jahren stark an, was auch heute noch anhält.
Es gibt 2 grundlegende Systeme um den Fehlern „Herr zu werden".

1) Automatic Repeat Request (ARQ)
Wie der Name schon sagt, handelt es sich hierbei um eine Anfrage auf erneute Sendung der Information.
Der Empfänger braucht nur ein System zur Fehlererkennung.
Erkennt er einen Fehler, sendet er ein NAK (negative acknowledgment) an den Sender und dieser Sendet die Folge nocheinmal.
Vorteil : Decodierung geht schneller und ist einfacher zu realisieren
Nachteil : ARQ stellt eine Extrabelastung für den Sender dar, besonders wenn ein Kanal sehr fehleranfällig ist.
Für grosse Distanzen nicht einsetzbar (z.B. Weltraumkommunikation)

2) Forward Error Correction (FEC)
Fehlererkennung und Korrektur mit Hilfe von Fehlerkorrigierenden Codes, worauf ich im folgenden näher eingehen werde.
Forward hat hier allerdings nicht damit zu tun, in welcher Richtung ein Code bei Codierung / Decodierung abgearbeitet wird, sondern bedeudet, das ein Fehler zuerst erkannt werden muss, um korrigiert zu werden.

Blockcodes

Ein Code ist eine bijektive Funktion, die jedem Wort aus S, Sd{BHBH ... HB}, B={0;1} ein Wort aus C, Cd{BHBH ... HB} zuordnet.
Man erzeugt einen (n,k) Blockcode, indem man zu den k Informationsbits, n-k redundante Bits hinzufügt. Man kann das Codewort demnach in 2 Teile zerlegen, den Informationsteil, der k Bits enthält, und einen Codeteil, der aus n-k Bits besteht, die aus dem Informationsteil berechnet werden. Dieser Codeaufbau wird häufig auch als systematisch bezeichnet.

Beispiel einer Fehlerkorrektur
Als Code habe ich den (5,2) Code verwendet, den ich gerade gezeigt habe.
Angenommen jemand möchte die Information 01 verschicken, entwickelt der Codierer aus der Folge das Codewort 01011. Beim Senden dieses Codewortes durch einen Kanal wird das Codewort aber verändert. Ich habe im Folgenden einen 1 Bit und einen 2 Bit Fehler produziert.
Der Decoder schaut sich jetzt die Hammingdistanzen der empfangenen Worte zu allen Codewörtern an. Das Codewort, was den geringsten Abstand zu dem empfangenen Wort aufweist, gibt der Decoder dann als Information aus.
Unser Code korrigiert dadurch einen 1 Bit Fehler problemlos. Als Ergebnis wird 01 berechnet.
Bei 2 oder Mehrbitfehlern wird eine andere Information decodiert. In unsrem Fall 00 statt 01.

Betrachten wir dazu die Korrekturfähigkeit von Blockcodes

Die Korrekturfähigkeit eines Blockcodes ist abhängig von der minimalen Distanz zwischen 2 Codewörtern d_{min}.
Es gilt : dmin = s + t + 1, wobei s die Anzahl der erkannten und t die Anzahl der korrigierten Fehler darstellt, wobei t immer kleiner als s ist, da nur ein erkannter Fehler auch korrigiert werden kann. Ein Fehler vom Ausmaß dmin wird nicht-korrigierbarer Fehler genannt, da es wieder ein Codewort ist.
Für d_{min} = 5 sind folgende (s,t)-Tupel möglich : (4,0);(3,1);(2,2)
Bei unserem Code war d_{min} = 3 -> (s,t) : (2,0) oder (1,1)
Dieser Code kann demnach 2 Bitfehler erkennen, aber nicht korrekt korrigieren.
1 Bitfehler jedoch korrigiert er problemlos.

Nun zum Hauptthema, den linearen Codes.
Die Einführung der Linearität bei Blockcodes kann man über die Galois-Felder gehen, wobei ich nur ein wenig an der Oberfläche kratzen werde.
Ein Galois-Feld ist eine Menge von Werten mit 2 darauf definierten binären Operationen „+“ und „x“, sowie deren Inversen „-“ und „/“, die nur Ergebnisse hervorbringen, die innerhalb des Feldes liegen. Mathematisch gesehen ist das das gleiche wie ein Körper.
An einen linearen Blockcode werden folgende Forderungen gestellt :
1. C(x1) + C(x2) = C(x1 + x2)
2. C(8x) = 8C(x)
3. Das Enthaltensein des Nullwortes als Codwort

Dabei sind diese Operationenist als Operationen modulo 2 eingeführt, damit man den Wertebereich dieses Feldes nicht verlässt.

Widmen wir uns nun ein wenig den Möglichkeiten der Produktion eines linearen Codes.
Dies ist über sogenannte Paritätskontrollen möglich, d.h. man bildet die Summe der Informationsbits und setzt danach das Paritätsbit.
Besitzt das Codewort eine gerade Anzahl von Einsen spricht man von gerader Parität.
Besitzt das Codewort eine ungerade Anzahl von Einsen spricht man von ungerader Parität.
Wie euch sicher schon aufgefallen ist, enthält der Code mit ungerader Parität nicht das Nullwort. Wir haben damit einen nicht linearen Code produziert. Nur der Code mit gerader Parität ist linear.
Systeme, die gerade Paritätskontrollen verwenden, sind automatisch linear.

Die meisten angegeben Codes haben die Eigenschaft, das die Informationsbits unverändert wieder im Codewort auftauchen. Man nennt diese Codes systematisch. Jeder lineare Code kann in eine systematische Form gebracht werden.

Als Konsequenz der Linearität ergibt sich eine gleichbleibende Distanz zwischen Codewörtern.
Die minimale Distanz eines linearen Blockcodes ist gleich der Mindestzahl der Symbole ungleich 0, die in jedem Codewort (ausser dem Nullwort) auftauchen.
Die Zahl der Symbole ungleich 0 in einer Folge wird auch Gewicht genannt, d.h. die minimale Distanz einen linearen Blockcodes ist durch das Gewicht des Codewortes gegeben, welchen das geringste Gewicht hat.

In den vorangegangenen Beispielen wurden die Codewörter an Hand einer Tabelle entnommen. Allerdings kann man die Linearität ausnutzen, um eine besser Möglichkeit zu finden, die sich auch besser umsetzen lässt. Da man durch Addition von Codewörtern immer auf ein Codewort kommt, benötigt man k linear unabhängige Codewörter, um jede Folge von Informationsbits herzustellen und damit auch jeden Code.

Sinnvollerweise nimmt man die Codewörter, deren Informationsteil ein Einheitsvektor ist. Die entstehende Generatormatrix besteht aus einer kxk Einheitsmatrix gefolgt von einer kx(n-k) Paritätskontrollbitmatrix. Bei einem systematischen Code benötigt man nur die Paritätskontrollbitmatrix, da die Information unverändert im Codewort wieder auftaucht.

Wie wir wissen, erzeugen gerade Paritäten einen linearen Code. Daher sollte es möglich sein einen Code in Bezug auf Bitgruppen zu definieren, die eine gerade Parität haben.
Man kann z.B. den (7,4) Blockcode wie folgt einteilen : PP Folie
So kann sichergestellt werden, dass der Code eine gerade Parität hat.
Die Matrix H wird Paritätskontrollmatrix genannt. Jede Zeile repräsentiert eine gerade Paritätsgruppe mit Einsen in den Positionen der Bits, welche die geraden Gruppen bilden.
Man kann die Matrix auch ausgehend von der Generatormatrix wie folgt bilden : PP Folie
Da jede Zeile von H mit einer geraden Paritätsgruppe korrespondiert, ist das Skalarprodukt eines jeden Codewortes mit einer beliebigen Zeile von H Null. Da die Zeilen der Generatormatrix selbst Codewörter sind gilt $GH^T=0$.

Wenn man nun einee Folge empfängt, ist es nun nicht wirklich effizient, die Folge mit allen Codewörtern zu vergleichen. Man bildet das Syndrom dieser Folge, indem man die Folge mit der transponierten Paritätskontrollmatrix multipliziert.
Ist v ein Codewort, so ist das Syndrom 0. Sobald ein Syndrombit 1 ist, ist die empfangene Folge fehlerhaft. Dann muss e so bestimmt werden, das v+e ein Codewort ist. => PP Folie

Eine weitere Möglichkeit den Decodierungsprozess zu veranschaulichen ist das logische Standardfeld, einer Liste aller empfangener Bitfolgen. Die Folgen werden dabei in Gruppen eingeteilt, denen ein Codewort zugeordnet wird. Bei einem (n,k) Code gibt es 2^k Codewörter und 2_n Bitfolgen. Man teilt diese in 2^{n-k} Gruppen ein. Dabei stehen oben immer die Codewörter und darunter die zugeordneten Bitfolgen. Dabei ist in einer Zeile das Syndrom gleich. Die letzten 2 Zeilen in unserem Standardfeld wurden willkürlich ausgewählt und sind für diesen Code nicht korrigierbar. Ein Decodiere würde wohl so konzipiert sein, dass er diese Fehler nur erkennt und nichts zur Korrektur unternimmt. Man nennt ihn dann einen unvollständigen Decodierer.
Die erste Folge einer Zeile wird als Errorvector behandelt und als Generator auf die erste Zeile angewendet. Die Element einer Zeile werden Restklassen genannt, der Errorvector auch Restklassenrepräsentant. Diese Restklassenrepräsentanten sollten so gewählt werden, das sie ein minimales Gewicht haben. Standardmäßig sind das alle Errorvectors ausgehend vom Nullwort. Damit kann dann eine Decodierung nach minimaler Distanz gewährleistet werden.
Zur Decodierung wird dann eine Tabelle erstellt, in der Syndrome und zugehörige, warscheinlichste Errorvectoren stehen. So wird eine Folge dann decodiert.

Desweiternen möchte ich noch ein paar Spezielle Codes vorstellen.
Hammingcodes
PP Folie
Zwar sind Hammingcodes immer vollständig und perfekt, was ich später noch erkläre, allerdings sind sie nur zur Einbitfehlerkorrektur fähig und aufgrund ihres begrenzten Variablensatzen der Länge n und Dimension k nur begrenzt einsetzbar.

Erweiterte Codes
Einen Code erweitern bedeutet, dass man extra Paritätskontrolbits anhängt, d.h. man erhöht n bei gleichbleibenden k. Bei einer ungeraden minimalen Distanz bewirkt dies eine Erhöung dieser um 1.

Verkürzte Codes

Einen Code verkürzen heißt, die Zahl der Informationsbits bei gleichbleibender Anzahl der Paritätskontrollbits zu verringern. N und k verringern sich somit um den gleichen Betrag.
Nehmen wir den (7,4) Hammingcode. Setzt man z.B. das 3.te Informationsbit 0, braucht man die 3 Zeile der Generatormatrix nicht mehr zu betrachten. Das Entfernen des 3. Bits resultiert dann im Entfernen der 3. Spalte. Somit ist auch wieder eine Einheitsmatrix hergestellt.
Die Paritätskontrollen am Ende der 3. Zeile tauchen auch in der Paritätskontrollmatrix auf. Da muss auch noch die 3. Spalte gelöscht werden. Man hat nun einen (6,3) Code erzeugt.
Veringern kann sich die Distanz dabei nicht, aber sie erhöht sich nicht automatisch. Dies muss im speziellen immer neu betrachtet werden und ist nicht allgemein formulierbar.

Grenzen von Blockcodes

Die Hamming-Grenze besagt, dass die Zahl der Syndrome mindestens gleich hoch ist, wie die Zahl der korrigierbaren Fehlermuster. Ein q-wertiges Symbol kann q-1 Fehler annehmen.
Daraus ergibts sich folgende Gleichung. Ergibt sich Gleichheit, wird der Code perfekt genannt, was aber nur bei Hammingcodes und dem Golaycode zutrifft.

Die Plotkin-Grenze legt einen maximal erreichbaren Wert für d_{min} für feststehende Werte für n und k fest.
Sie sagt aus, dass die minimale Distanz bei linearen Codes maximal gleich hoch ist, wie das gemittelte Gewicht aller Codefolgen außer dem Nullwort. Die Warscheinlichkeit eines Symbols ungleich Null ist (q-1)/q und es existieren q^k Codewörter. Die Zahl der Codewörter ungleich Null ist q^k -1. Daraus ergibt sich dann die Gleichung. Die minimale Distanz kann diesen Wert nicht übersteigen.

Diese Formel lässt sich auch so umformen, dass man eine Grenze für k bei gegebenen n und d_{min} erhält.

Die Griesmer-Grenze ist meist niedriger als die Plotkin-Grenze und es können aus ihr Methoden zur Entwicklung guter Codes abgeleitet werden. N(k,d) repräsentiert das niedrigste n für einen linearen Code der Dimension k und der minimalen Distanz d.
Dieses n erfüllt auch wieder die Plotkin-Grenze mit Gleichheit.

Singleton-Grenze
Wird eines der Informationsbits verändert, so is das beste was man hoffen kann, dass sich auch alle Paritätsbits ändern. In diesem Falle wäre die Distanz zwischen 2 Codewörtern n-k+1, was wiederum eine Obergrenze darstellt. Allerdings erreichen nur sog. Wiederholungscodes und Reed-Solomoncodes diese Obergrenze.

Es gibt noch weiter Grenzen für lineare Code wie die Kugelpackungsgrenze oder die Gilbert-Varsharmov-Grenze, aber das würde jetzt zu weit führen.

Ich hoffe ich konnte einen kleinen Einblick in das Gebiet dieser Art der Fehlerkorrektur geben und beende damit meinen Vortrag.