Kanalkodierung

Leistungsnachweis
Wintersemester 2009/2010

Christian Schubert
3234139

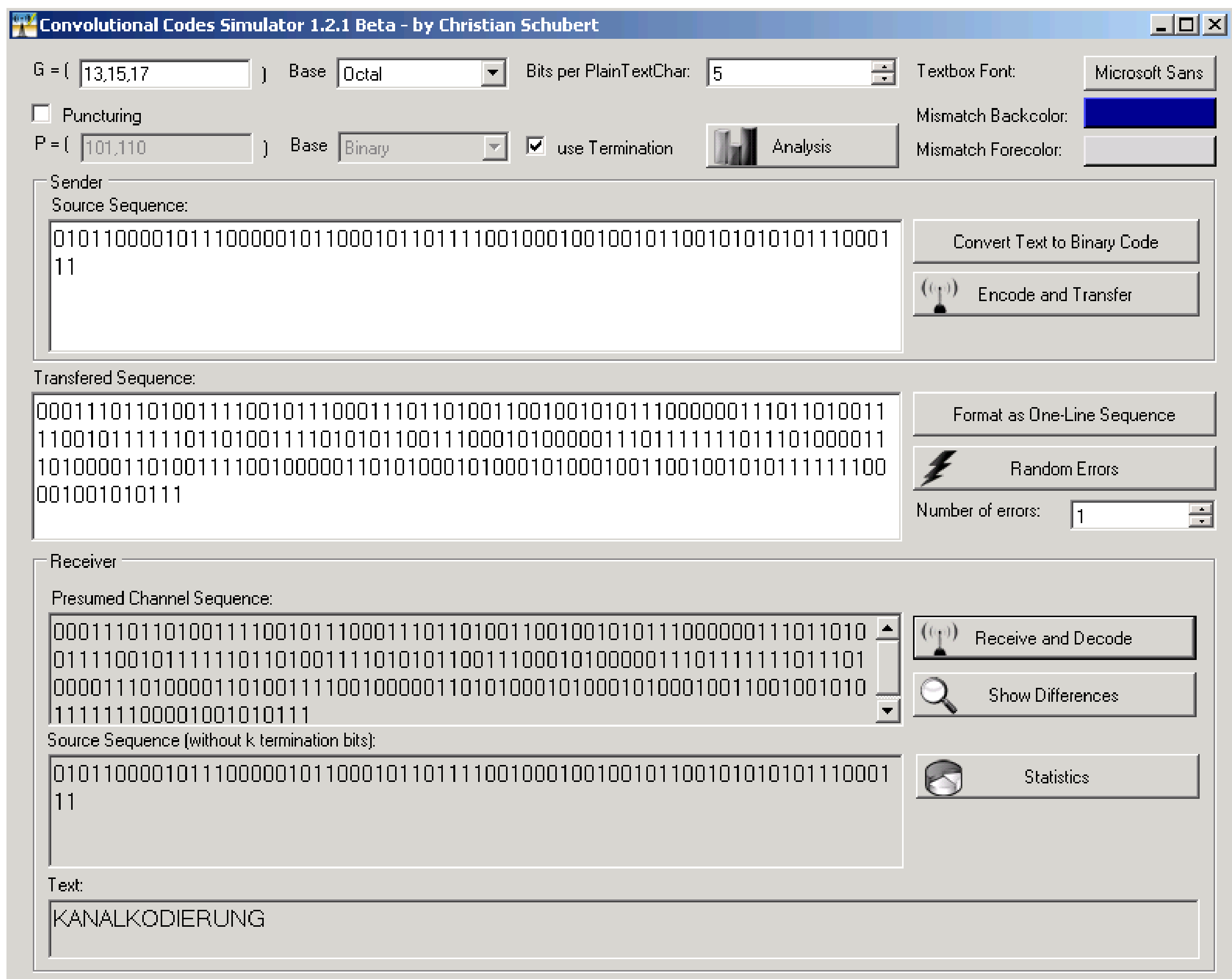
1 Convolutional Codes Simulator: Ein alternativer Faltungskodierer/-dekodierer

Diese hier vorliegende Arbeit wurde in fachlich enger Kooperation mit den beiden Studenten Rico Pöhland und Mandy Korzetz entwickelt, wobei die Implementierung getrennt verlief und dadurch beide Arbeiten jeweils, für sich gesehen, einen vollwertigen Beitrag zur Thematik Faltungskodes darstellen.
Bedingt dadurch, bezieht sich der hier vorliegende Text auch in erster Linie auf die entwickelte Anwendung mitsamt ihrer Bedienung, den internen Ablauf und den direkten Vergleich mit jener Applikation, welche durch M. Korzetz und R. Pöhland entwickelt wurde.
Da beide Programme grundlegend das Gleiche leisten und ohnehin eine Kooperation bestand, wurde auf die weiteren Faltungskode-Aufgaben des Leistungsscheins nicht weiter eingegangen und stattdessen am Ende neben dem genannten Vergleich eine Untersuchung bzgl. Punktierung vorgenommen.

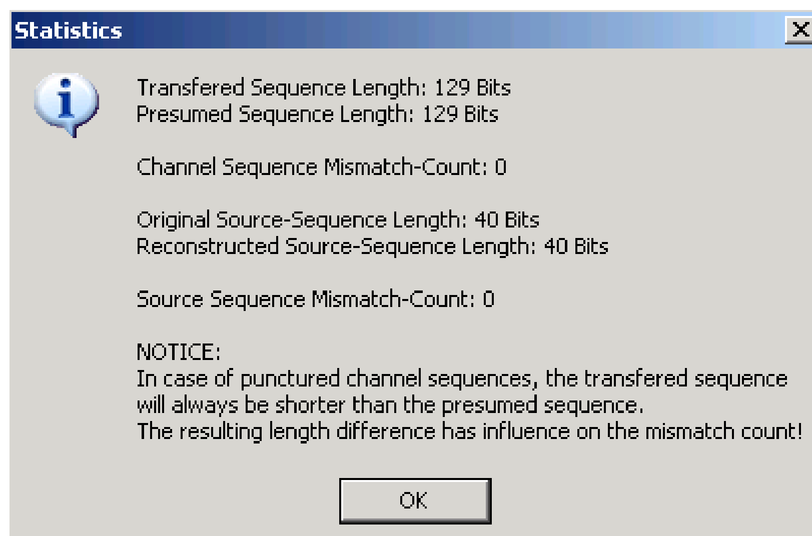
2 Benutzeroberfläche

Der Convolutional Codes Simulator(kurz: CCS) bietet die Möglichkeit, den Prozess des Kodierens, Übertragens(optional mit Fehlern) und Dekodierens schrittweise nachzuvollziehen.
Um die genannten Funktionen möglichst intuitiv dem Bediener verfügbar zu machen, wurde die grafische Bedienoberfläche in entsprechende Bereiche gegliedert sowie auf das Hauptfenster und das Fehleranalysefenster aufgeteilt.

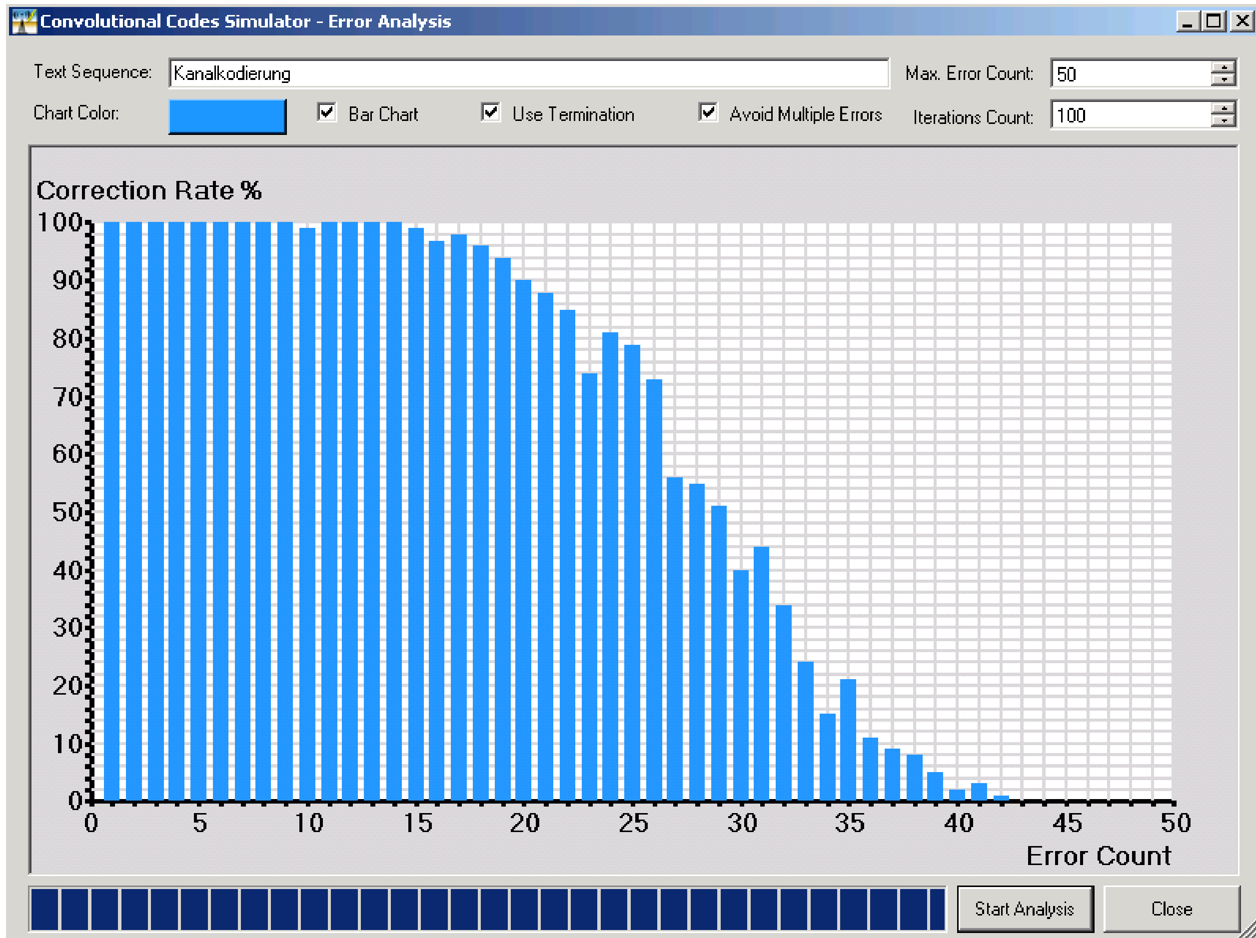
2.1 Das Hauptfenster



Der Aufbau des Hauptfensters entspricht grundlegend dem Ablauf, wie er bei dem Prozess Kodieren → Übermitteln → Dekodieren vorliegt, wobei die Anordnung der jeweiligen Abschnitte der vertikalen Richtung von Oben nach Unten entspricht.
Im obersten Bereich befindet sich der Definitionsbereich mit den benötigten Eingabefeldern für den gewünschten Faltungskode. Hier hat der Nutzer die Möglichkeit der Angabe der Generatormatrix G, die optionalen Punktierungsmatrix P sowie weiterer Parameter (Terminierung, Zeilendefinitionsbasis). Zusätzlich gibt es die Option, die verwendete Text-Schriftart und Farben den persönlichen Bedürfnissen anzupassen.
Um eine Matrix zu definieren, ist zu beachten, dass die Eingabe zeilenorientiert abläuft und die Matrixzeilen durch Kommas getrennt werden müssen. Desweiteren muss die Eingabe immer entsprechend der jeweiligen Zahlenbasis geschehen, wobei hier zu Auswahl "Oktal", "Hexadezimal", "Dezimal" und "Dual" stehen.
Als Letztes sei der Schalter "Analysis" erwähnt, welcher den Zugriff auf die Fehlerfallanalyse ermöglicht.
Im Sender-Bereich finden alle Aktionen statt, welche das Konvertieren von Text in Quellkodewörter sowie das Kodieren eben jener zu Kanalkodewörter betrifft. Das Eingabefeld kann mit binären Quellkodewörtern oder auch Text(A – Z und Leerzeichen) gefüllt werden, wobei aber gemischte Sequenzen nicht möglich sind. Zusätzlich muss nach der Texteingabe eine Konvertierung in eine binäre Quellkodesequenz durchgeführt werden. Danach ist das Kodieren und Senden der Kanalkodesequenz möglich.
Im mittleren Bereich des Hauptfensters befindet sich ein Eingabefeld, was die momentane "gesendete" Kanalkodesequenz enthält. Diese kann durch manuelle Modifikationen oder der Einstreuung von Fehlern manipuliert werden, sodass diverse Fehlerszenarien möglich sind.
Zur Nutzung der Zufallsfehlereinstreuung befinden sich am rechten Rand ein Schalter "Random Errors" und ein Nummernfeld, wo die Anzahl der Fehler eingestellt werden kann.
Im darunter befindlichen Empfänger-Bereich findet primär die Dekodierung sowie die Differenzbetrachtung zwischen gesendeter und empfangener Sequenz statt. Auf der linken Seite befinden sich die Felder für die Ausgabe der angenommenen Kanalkodesequenz, der dekodierten Quellkodesequenz sowie der darauf basierenden Textdarstellung.
Auf der rechten Seite befinden sich neben der Schaltfläche für die Dekodierung zwei weitere für die Differenzbetrachtung sowie die Statistikanzeige. Bei der Anzeige der Differenzen werden die gesendete(sofern vorhanden) und empfangene Quellkodesequenz sowie die übertragene und die angenommene Kanalkodesequenz miteinander verglichen und danach optisch hervor gehoben. Die Statistikanzeige selbst gibt in kompakter Form Auskunft über alle gesendeten wie auch empfangenen Sequenzen sowie fehlende Übereinstimmungen.



2.2 Das Fehleranalysefenster



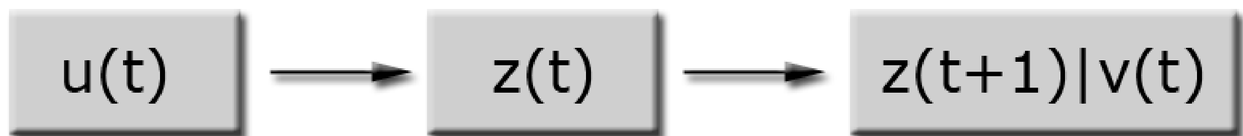
Dieser Teil der Applikation erlaubt die empirische Fehleranalyse von Faltungskodes und somit entsprechende Aussagen über die Leistungsfähigkeit eben jener.
Im oberen Bereich befinden sich das Texteingabefeld(A-Z sowie Leerzeichen), numerische Felder für die Anzahl der maximal einzufügenden Fehler und der Anzahl Iterationen pro Fehlerstufe sowie diverse Schalter, um den Ablauf des Testlaufes bzw. dessen Visualisierung zu modifizieren.
Im mittleren Bereich, welcher den größten Teil der Gesamtfläche einnimmt, befindet sich das Diagrammfeld, welches zur Darstellung der Testergebnisse benötigt wird. Nach dem erfolgreichem Ablauf eines Testlaufes werden die Ergebnisse darin per Balkendiagramm dargestellt.
Direkt unter dem Diagrammfeld befindet sich ein Fortschrittsbalken sowie zwei Schaltflächen zum Starten des Testlaufes respektive Schließen des Analysefensters.
Zu beachten ist, dass, abgesehen von den möglichen Änderungen im Analysefenster(z.B. Terminierung an/aus,..), alle anderen Einstellungen für den Faltungskodierer/-dekodierer direkt vom Hauptfenster der Applikation übernommen werden, sodass direkte Änderungen nur dort möglich sind.

3 Interne Verarbeitung

3.1 Allgemeine interne Vorbereitungen

Sowohl für die Kodierung als auch die Dekodierung ist es notwendig, dass mindestens die Generatormatrix G vollständig und korrekt beschrieben ist. Hierfür wird der Inhalt des jeweiligen Eingabefelds eingelesen und mit einem internen Parser auf Korrektheit der Formatierung sowie der

Beschreibung relativ zur verwendeten numerischen Basis geprüft. Falls hierbei eine Fehleingabe aufgedeckt wird, wird der Nutzer umgehend darüber informiert und der jeweilige Prozess abgebrochen. Nachdem diese Überprüfung erfolgreich abgeschlossen wurde, wird die binäre Generatormatrix auf Basis der gegeben Eingabe erstellt und zusätzlich ein Dimensionstest durchgeführt, damit sichergestellt ist, dass alle Zeilenvektoren gleichlang sind und somit die Form der verwendeten Matrix G korrekt ist.
Analog dazu verläuft die Generierung der optionalen Punktierungsmatrix P, da diese genauso beschrieben werden kann wie G. Auch hier kommt es zur Prüfung und Konvertierung mittels eines Parsers sowie der anschließenden Dimensionskontrolle.
Zusätzlich zu diesen beiden Komponenten für die Faltungskodierung/-dekodierung ist die Angabe, ob Terminierung verwendet wird, ausschlaggebend und essentiell.
Nachdem das System die genannten Eingaben intern übernommen hat, beginnt die effektive Vorbereitung für die spätere Kodierung/Dekodierung.
Zunächst werden essentielle Variablen(k,m,..) anhand der gegeben Eingaben berechnet respektive extrahiert und anschließend gespeichert. Desweiteren wird hier nun die Grundlage für den später angewandten Viterbi-Algorithmus berechnet. Hierbei handelt es sich um eine zweistufige Lookup-Tabelle, welche in ihrer Funktion die bekannte Transitionstabelle repräsentiert und zugleich in abstrakter Form das verkürzte Trellis-Diagramm widerspiegelt.
Der Aufbau hierfür ist analog zur bekannten Transitionstabelle bei Faltungskodes und unterscheidet sich nur unwesentlich von eben jener(Implementierungsdetails ausgenommen).
Folgende Darstellung soll ihren Aufbau und Funktion verdeutlichen:



Wie man sehen kann, wird zunächst ein Quellkodebit u (0 oder 1) zu einem beliebigen Zeitpunkt t benötigt, um dadurch die erste Stufe zu passieren.Als Nächstes wird ein Speicherzustand(auch Gedächtniszustand genannt) z zum genannten Zeitpunkt t benötigt. Nachdem nun diese Angabe gemacht und die zweite Stufe passiert wurde, bekommt man als Ergebnis sowohl den nachfolgenden Zustand z(t+1) als auch die resultierende Kanalkodebitfolge v(t).
Bei der programmtechnischen Realisierung wurde für den Aufbau der LookUp-Tabelle ein rekursiver Ansatz gewählt, um damit alle möglichen Speicherbelegungspermutationen schnell und vollständig zu realisieren.
Weitere Vorverarbeitungsschritte finden nicht statt, d.h. im Gegensatz zu anderen Applikationen, welche ein vollständiges Trellis-Diagramm benötigen, ist dies hier nicht notwendig, da sowohl die Kodierung als auch die Dekodierung, angepasst an dieser Struktur, arbeiten.

3.2 Kodierung

Zu Beginn wird geprüft, ob es sich bei der Eingabe um eine binäre Sequenz handelt und bei gegebener Nichterfüllung dieser Bedingung der Nutzer informiert.
Danach wird geprüft, ob Terminierung verwendet werden soll. Falls dem so ist, werden k Nullen an die gegebene Quellkodesequenz angehängt.
Im nachfolgenden Schritt kommt es zur eigentlichen Kanalkodierung, was sich dank der vor-berechneten LookUp-Tabelle als sehr einfach und schnell gestaltet.
In einer For-Schleife werden alle Quellkodebits sequentiell abgearbeitet, wobei parallel dazu immer

ein sog. State als Variable gehalten und aktualisiert wird. Dieser stellt laut obiger Darstellung das z(t) dar und wird nach erfolgter Verarbeitung des aktuellen Quellkodebits u(t) auf z(t+1) gesetzt.
Die Kanalkodesequenz selbst wird dadurch gebildet, dass alle v(t), welche sich beim Durchlaufen der Schleife ergeben, aneinandergereiht werden.
In Folge kommt es zur optionalen Punktierung, welche sich durch wiederholtes Durchlaufen der Periode der Punktierungsmatrix P analog zum dem von Faltungskodes bekannten Verfahren handelt. Auch hier gilt, dass nur bei entsprechender "1" an der aktuellen Stelle in P das aktuelle Kanalkodebit erhalten bleibt.
Nachdem dieser Vorgang beendet wurde, wird die resultierende Kanalkodesequenz im entsprechenden Feld ausgegeben und somit "gesendet".

3.3 Dekodierung

Die Dekodierung verläuft grundsätzlich identisch zum bekannten Viterbi-Algorithmus, wobei in diesem Fall die sog. Hard-Decision MD-Metrik verwendet wird.
Zu Beginn wird, analog zur Kodierung, die Eingabe, d.h. die Kanalkodesequenz bzgl. erlaubter Symbole validiert und ggf. der Nutzer bei Verletzungen der Vorgabe(nur binäre Sequenzen erlaubt) darüber informiert.
Falls Punktierung zum Einsatz kommt, wird anhand der gegebenen Punktierungsmatrix P die Depunktierung der Eingabesequenz durchgeführt, wobei hier als Dont-Care Symbol das Zeichen "#" fungiert.
Danach beginnt die eigentliche Abarbeitung des Viterbi-Algorithmus, indem zunächst die Anzahl an notwendigen Schritten berechnet wird, welche sich durch die Länge des Kanalkodesequenz, dividiert durch m ergibt. Dies ist auch schon eine einfache Möglichkeit, Unstimmigkeiten in der empfangenen Sequenz zu ermitteln, denn grundsätzlich gilt, dass m Teiler der Sequenzlänge sein muss.
Um die Zwischenergebnisse der Knoten des Viterbi-Algorithmus speichern zu können und später somit ein effektives Backtracking zu ermöglichen, wird ein Array von Listen angelegt, was die gesamte Länge der Eingabesequenz abdeckt.
Für den Viterbi-Algorithmus dient eine For-Schleife als äußere Kontrollstruktur, um die gegebene Sequenz schrittweise zu Durchlaufen, wobei pro Schritt genau m sequentiell angeordnete Bits verwendet werden, um die Metrik auf Basis der Hamming-Distanz zu berechnen.
Da nur mit der oben beschriebenen Transitionstabelle gearbeitet wird, muss in gewisser Weise eine vorwärts gerichtete Abarbeitung stattfinden. D.h. pro Schritt werden alle erlaubten Quellkodebits als Keys für die 1. Stufe der LookUp-Tabelle verwendet, was bei $t < l$ immer Null und Eins sein kann, aber ab $t >= l$ nur noch Null ist, weil in jenem Fall definitiv Terminierung verwendet werden würde (falls dem nicht so ist, ist der Algorithmus ab $t = l$ sowieso fertig).
Direkt im Anschluss findet iterativ die Berechnung aller Distanzen sowie der kumulierten Metrikenwerte zum aktuellen Zeitpunkt t ausgehend von den Viterbi-Knoten zum Zeitpunkt t – 1statt. Programmiertechnisch wird dies durch eine Iteration durch die vorherige Liste(bezogen auf t) realisiert.
Da immer ein Viterbi-Knoten durch zwei Transitionen erreicht wird(welche bekanntlich die Quellkodebits Null und Eins repräsentieren) und in diesem Fall das MD-Prinzip verwendet wird, wird nach jeder Metrikberechnung geprüft, ob die aktuelle Transition insgesamt optimaler ist , d.h. ein niedrigerer kumulierter Metrikwert vorliegt, als die bisherige optimale Transition(sofern vorhanden) und jene ggf. ersetzt. Dieser Vorgang entspricht dem Streichen von nicht optimalen Transitionen beim Viterbi-Algorithmus.
Nach erfolgreicher Abarbeitung der gesamten Kanalkodesequenz wird nun das sog. Backtracking angewendet, bei dem, ausgehend vom Endknoten, der optimale Weg rückwärts durch den Transitions-Graph durchwandert wird. Hierbei werden alle Transition, welche verwendet werden, in

einer dynamische Liste gespeichert und am Ende diese Liste bzgl. ihrer Reihenfolge umgekehrt. Dadurch wird sowohl die ermittelte Quellkodesequenz als auch die angenommene Kanalkodesequenz generiert und anschließend ausgegeben, wobei im Falle von Terminierung vor der Ausgabe jedoch die Terminierungsbits entfernt werden.
Zusätzlich dazu wird nachfolgend die Konvertierung der Quellkodebits, welche auf einem vorberechneten Alphabet basiert, in alphanumerische Zeichen vorgenommen.

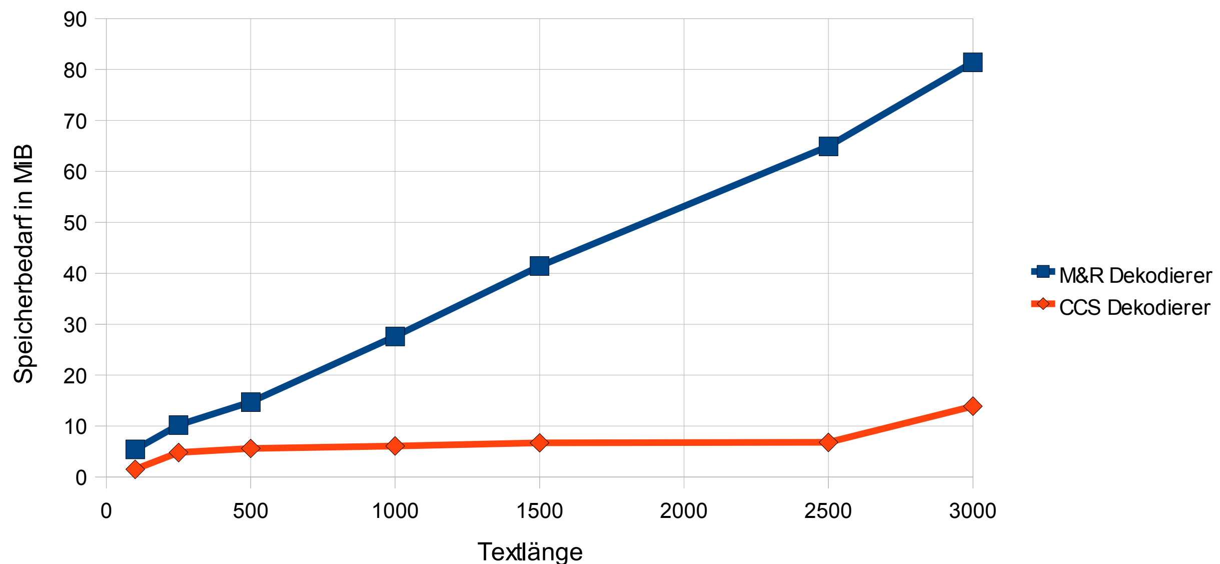
3.4 Fehlerfallanalyse

Die Fehlerfallanalysefunktion, welche im entsprechenden Bereich der Applikation angeboten wird, verwendet die bereits erwähnten Kodierungs- und Dekodierungsfunktionen des Programms und basiert grundlegend auf dem Vergleich der ursprünglich berechneten Kanalkodesequenz und der angenommene Kanalkodesequenz nach der Dekodierung.
Iterativ werden alle Fehlerstufen bis zum vorgegeben Maximum durchlaufen(wobei jede Stufe immer auch die Anzahl an Fehlern darstellt) und pro Stufe wiederum iterativ, basierend auf der vorgegeben Anzahl von Iterationen pro Fehlerstufe, eine durchschnittliche Korrekturrate ermittelt, indem jedesmal die ursprüngliche Kanalkodesequenz kopiert, jene Kopie mit zufälligen Fehler modifiziert und danach diese dekodiert wird.
Die dabei entstandene angenommene Kanalkodesequenz wird anschließend mit der Original-Sequenz, wie bereits erwähnt, auf Übereinstimmung verglichen.
Nach jeder Fehlerstufe wird so die Korrekturrate arithmetisch gemittelt und für die spätere Präsentation zwischengespeichert.

4 Untersuchungen

4.1 Vergleich bezüglich Speicherverbrauch

Wie bereits weiter oben erwähnt, wurde der hier besprochen Faltungskodierer/-dekodierer parallel zu dem von M. Korzetz und R. Pöhland entwickelten Programm entwurfen und entwickelt.
Beide Programme leisten grundsätzlich das Gleiche, verwenden aber verschiedene Ansätze um Kodierung bzw. Dekodierung von Faltungskodesequenzen zu realisieren.
Während der hier besprochene Faltungskodierer/-dekodierer auf Basis einer zweistufigen LookUp-Table, welche als verkürztes Trellis-Diagramm fungiert, arbeitet, nutzt jener von M. Korzetz und R. Pöhland das voll aufgespannte Trellis-Diagramm, welches bekanntlich direkt von der Länge der Kanalkodesequenz abhängig ist.
Beide Ansätze sind möglich und arbeiten korrekt, sodass hier keine allgemeine Bewertung bzgl. “richtig“ oder „falsch“ getroffen werden kann und soll. Je nach Szenario und technischen Randbedingungen ist dies nur im Einzelfall möglich, weshalb ich mich beim Vergleich hier auf die maximale Speicherauslastung als Kenngröße konzentriert habe..
Im konkreten Fall ging es um die maximale Speicherauslastung während der Dekodierung von Kodesequenzen. Dazu wurden mehrere Texte mit verschiedener Länge zunächst kodiert und danach durch die Dekodierer meiner Applikation(“CCS“) und auch jener von M. Korzetz und R. Pöhland(“M&R“) dekodiert. Hierbei wurden die Standard-Einstellungen(bei CCS und M&R identisch) verwendet und jeweils mehrere Durchläufe durchgeführt, wobei der Speicherverbrauch parallel beobachtet und die Peak-Werte für den Vergleich aufgenommen wurden. Im Folgenden sieht man die Ergebniswerte der Messungen sowohl tabellarisch als auch graphisch aufbereitet.



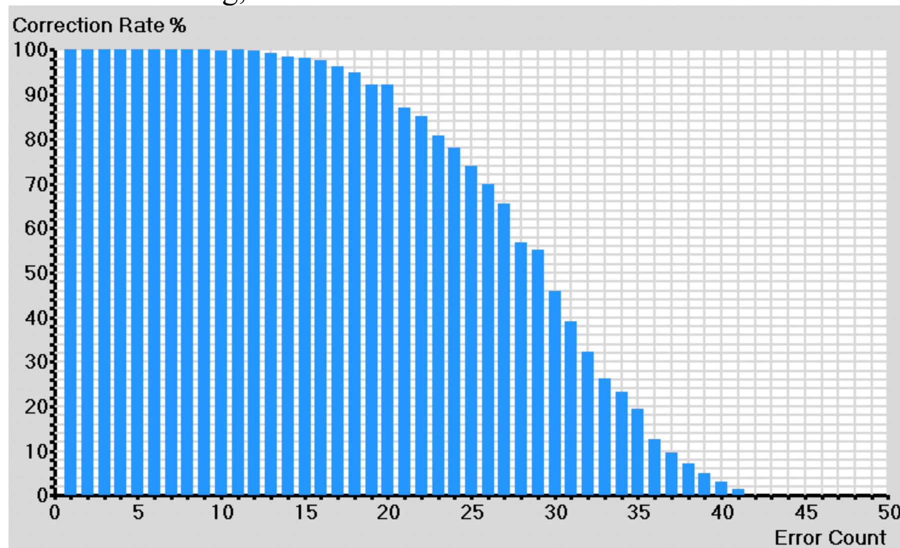
Originale Textlänge	CCS Dekodierer	M&R Dekodierer
100	1,5	5,4
250	4,8	10,2
500	5,6	14,7
1000	6,1	27,6
1500	6,7	41,4
2500	6,8	64,9
3000	13,9	81,4

Wie man anhand des Diagramms erkennen kann, verhalten sich beide Dekodierer sehr unterschiedlich bzgl. der Speicherauslastung. Während der M&R Dekodierer sich, abgesehen von einigen Unregelmäßigkeiten, nahezu linear bei der Speicherauslastung verhält, was sich wiederum mit dem oben beschriebenen Aufbau des kompletten Trellis-Diagramms und der internen Verarbeitung deckt, bleibt der CCS Dekodierer deutlich darunter und lässt anhand der gegeben Messwerte fast ein logarithmisches Verhalten erkennen, welches nur durch den letzten Messwert in seinem Verlauf gestört wird.
Was die kleineren Unregelmäßigkeiten angeht, muss hierbei erwähnt werden, dass sich .NET Applikationen, wie sie hier vorliegen, bei der Speichernutzung nie identisch verhalten, da die automatische Speicherverwaltung sich nach Außen indeterministisch verhält und jederzeit Änderungen durchführen kann. Aus diesem Grunde haben die erhobenen Werte auch keinerlei Anspruch auf absolute Korrektheit und dienen primär dem direkten Vergleich in diesem speziellen Fall und zur groben Orientierung.
Abschließend sei gesagt, dass, abgesehen vom besagten Unterschied bei der Speicherausnutzung, beide Dekodierer ihren Stärken und Schwächen haben. Zum Beispiel kann der M&R-Dekodierer ohne zusätzlichen Aufwand den kompletten Dekodier-Prozess visualisieren und ist somit wunderbar dafür geeignet, Interessierte schnell und einfach an das Thema Faltungskodes heranzuführen.
Der CCS-Dekodierer wurde wiederum unter der Prämisse entwickelt, möglichst schnell und speicherschonend die gestellten Aufgaben zu erfüllen und ist somit eher für Leute geeignet, welche sich bereits mit dieser Thematik beschäftigt haben.

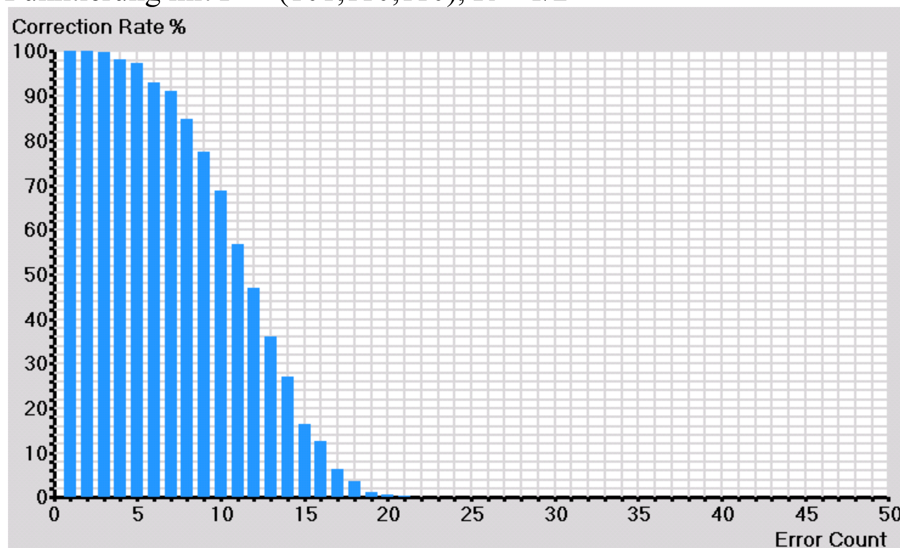
4.2 Einfluss von Punktierung

Da die Analysefunktion der Applikation auch die Punktierung berücksichtigt, war es naheliegend, den Einfluss von Punktierung auf die Leistungsfähigkeit eines Faltungskodes empirisch zu analysieren.
Hierfür wurden alle voreingestellten Parameter, abgesehen vom Aktivieren/Deaktivieren der Punktierung sowie Ändern der Punktierungsmatrix P, ohne Anpassungen verwendet und die Anzahl an Iterationen pro Fehlerstufe auf 1000 gesetzt, um damit möglichst repräsentative Werte ermitteln zu können, welche man den nachfolgenden Diagrammen entnehmen kann.
Zu beachten ist auch hier, dass die ermittelten Ergebnisse , trotz größter Aufmerksamkeit, keinerlei Anspruch auf absolute Korrektheit haben und auch nicht als Maßstab für weiterführende Verfahren genutzt werden sollten. Gründe hierfür liegen vorallem in der Tatsache begründet, dass nur zufällige Einzelfehler(ergeben u.U. Bündelfehler) eingesetzt sowie der vorgegebene Text verwendet wurde.

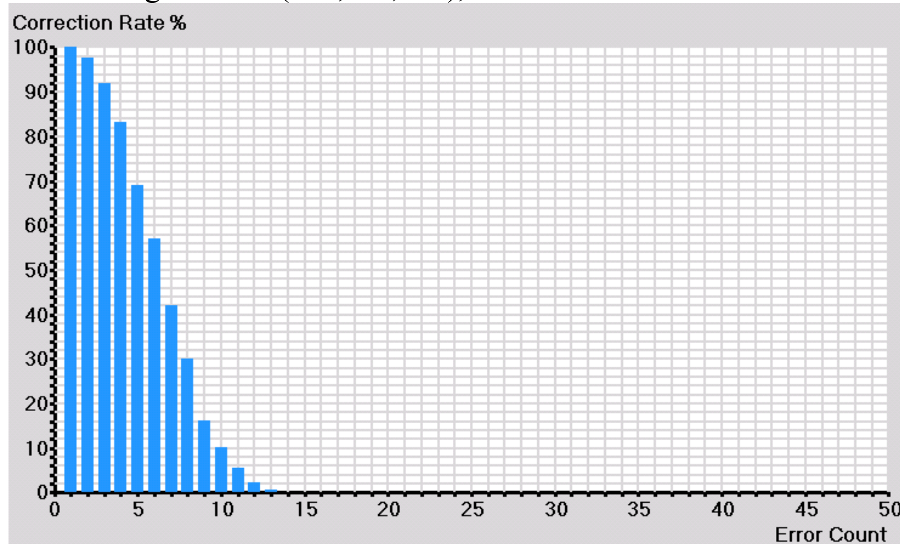
Ohne Punktierung, R = 1/3



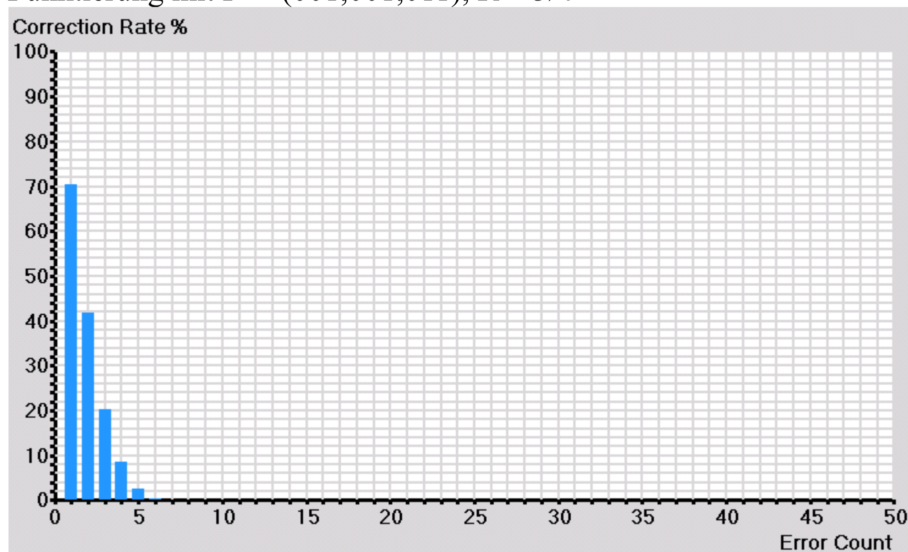
Punktierung mit P = (101,110,110), R = 1/2



Punktierung mit P = (001,111,001), R = 3/5



Punktierung mit P = (001,001,011), R = 3/4



Wie man sieht, verschlechtert sich die Leistungsfähigkeit eines gegeben Faltungskodes bzgl. Fehlerkorrektur abhängig von der verwendeten Punktierungsmatrix P teilweise dramatisch.
Gerade das letzte Beispiel zeigt, wie hoch der Preis für die um 125 % höhere Koderate, bezogen auf die ursprüngliche Korrekturrate, ist.
In diesem Fall wird selbst ein einzelner zufälliger Fehler im Durchschnitt nur in 70 % aller Fälle korrigiert und bereits ab vier Fehlern befindet sich die Korrekturrate im einstelligen prozentualen Bereich. Mehr als fünf Fehler können offensichtlich überhaupt nicht mehr korrigiert werden.
Zurückzuführen sind diese teilweise extremen Verschlechterungen auf die Tatsache, dass bei der Punktierung die besonders wichtige Redundanz sehr stark reduziert wird und dadurch zwangsweise im Falle von Fehlern diese zur Korrektur fehlt.
Bedingt durch diesen Umstand kam man auch hier feststellen, dass sich Koderate und Leistungsfähigkeit(konkret: die Minimaldistanz) konträr zueinander verhalten und dadurch auch keine gleichzeitige Maximierung beider Größen möglich ist. Aus diesem Grunde sollte Punktierung immer nur abhängig vom Szenario(Störverhalten,..) genutzt werden bzw. das komplette System(Kodierer,Dekodierer) adaptiv mit Punktierung arbeiten.