Notizen im Laufe der Bachelorarbeit

SOVA vom c-File:

Dabei wird der ML-Dekodierungs-Algorithmus von Buch Informations- und Kodierungstheorie verwendet.   
Dabei wird die Metrik mittels Skalarprodukt berechnet.   
Die Metrik muss maximiert werden.   
Der Eingang muss auf -1,+1 für die Berechnung gemapped werden.  
Fehlerwahrscheinlichkeit umso größer je kleiner Delta ist.

Log-Likelihood-Algebra vereinfacht Berechnung, Zuverlässigkeit von Bit ist der Betrag von L und die Hard-Decision ist ob L es positiv oder negativ ist.  
Berechnung der Zuverlässigkeit befindet sich im Buch auf Seite 231.  
  
Variablen der Formel:  
m…Anzahl Ausgängen am Kodierer  
k…Anzahl Register im Kodierer  
delte…Differenz der Metriken  
Metrik…Unterschied der Empfangsfolge zur möglichen Folge  
  
Erklärung von Bezeichnungen:  
systematisch -> Eingangszeichen liegen unverändert am Ausgang vor (Leitung von vorne muss durchgeschleift werden  
standartform -> alle Eingangszeichen sind am Anfang der Ausgangsfolge  
rekursiv-> eine Leitung wird von hinten zum Anfang gelegt  
freie Distanz->geringstes Gewicht(Anzahl von 1er) einer Ausgangsfolge, die von 0 zum 0 Zustand geht

Turbo-Codierer:  
Interleaver sorgt dafür, dass gewicht der Ausgangsfolgen steigt.

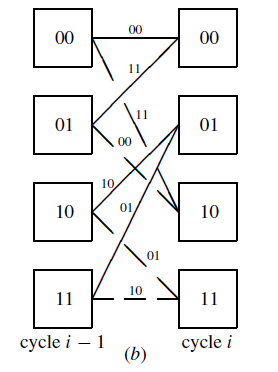
23/03/2016

Änderungen an SOVA Dekodierer um ihn passend für allgemeine Faltungskodes zu machen.  
Resultierendes C File: turbo\_map\_sova\_modified3.c

Grund für die Änderungen war die Verwendung der **previous** Matrix im Dekodiervorgang, die nur spezielle Faltungs-/Turbokodes zugelassen hat (siehe folgendes Beispiel)

Die previous Matrix speichert die Zustände ab, aus denen man mit einem bestimmten Inputbit in einen bestimmten Folgezustand gelangt. Für den bisher im Code verwendeten rekursiven Faltungskodierer sah diese Matrix folgendermaßen aus:  
Der Zeilenindex steht für den Folgezustand, der Spaltenindex für das Inputbit  
 \_\_\_\_\_\_\_  
|\_0\_|\_1\_|  
|\_3\_|\_2\_|  
|\_1\_|\_0\_|  
|\_2\_|\_3\_|

Für allgemeine Faltungskodierer kann jedoch folgender Fall eintreten, dass in einen bestimmten Zustand nur mithilfe **eines** bestimmten Inputbits gesprungen werden kann (siehe folgendes Beispiel – Standardfaltungskodierer, FEC S.552 Figure 14.2)



Hier kann bspw. in den Zustand 00 nur mithilfe eines Inputbits 0 gesprungen werden (entweder aus Zustand 00 oder aus Zustand 01).  
Dasselbe gilt für die anderen Zustände, jedoch u.U. mit einer 1 als Inputbits.

Daher kann die wie zuvor eingeführte previous Matrix nicht verwendet werden, da es bspw. für Zustand 00 und Inputbit 0 zwei mögliche ‚previous‘ Zustände gibt (nämlich 00 und 01), für Inputbit 1 jedoch **keinen**.

Daher wurde aus der 2-dimensionalen Matrix eine 3-dimensionale Matrix gemacht:

previous[2][#states][#inputs], wobei #inputs = 2 und (meist) #states = 4  
Die neue previous Matrix besteht aus 2 2-dim. Matrizen um alle möglichen previous states abzuspeichern, auch wenn previous states aus mit demselben Inputbit erreicht werden. Zellen der Matrix die nicht benötigt werden, werden mit dem Wert -1 belegt (für spätere Abfragen wichtig).

Für den „alten“ Kodierer sieht dann die previous Matrix dann folgendermaßen aus:  
 \_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_  
|\_\_0\_\_|\_\_1\_\_| |\_-1\_\_|\_-1\_\_|  
|\_\_3\_\_|\_\_2\_\_| |\_-1\_\_|\_-1\_\_|  
|\_\_1\_\_|\_\_0\_\_| |\_-1\_\_|\_-1\_\_|  
|\_\_2\_\_|\_\_3\_\_| |\_-1\_\_|\_-1\_\_|

Für den „Standard-Faltungskodierer“ sieht die previous Matrix folgendermaßen aus:  
 \_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_  
|\_\_0\_\_|\_-1\_\_| |\_\_1\_\_|\_-1\_\_|  
|\_\_2\_\_|\_-1\_\_| |\_\_3\_\_|\_-1\_\_|  
|\_-1\_\_|\_\_0\_\_| |\_-1\_\_|\_\_1\_\_|  
|\_-1\_\_|\_\_2\_\_| |\_-1\_\_|\_\_3\_\_|  
 „Matrix 0“ „Matrix 1“

Trifft man in der 1. Matrix (Index 0) auf eine „-1“ bedeutet das es gibt für diesen Zustand keinen previous state mit diesem Übergangsbit, d.h. um den zweiten möglichen previous state (in diesem Fall mit demselben bit) zu erhalten, muss in die zweite Matrix geschaut werden.

Wiederum gilt: Zeilenindex = Zustand/State, Spaltenindex = Inputbit {0,1}

Bei der alten Implementierung wurde bei der Metrik- und Deltaberechnung lediglich das survivor bit gespeichert. Dies genügt nun nicht mehr, da für ein bit evtl. 2 previous states in Frage kommen. Daher gibt es jetzt zusätzlich eine previous\_matrix\_decision[][] Matrix, die sich für jeden Zeitpunkt (1.Index) und jeden State (2.Index) merkt, ob die previous states über verschiedene Bits erreichbar sind (-1) und die „zweite“ Matrix der previous Matrix gar nicht benötigt wird, ODER in welcher Matrix der previous Matrix nachgeschaut werden muss, d.h. Matrix 0 oder Matrix 1.  
D.h. -1 signalisiert, dass verschiedene bits zu den previous states führen, hingegen eine 0 oder eine 1 signalisieren, dass gleiche bits zu den previous states führen. Ob Inputbit 0 oder Inputbit 1 bleibt im survivor bit gespeichert.

Kommentare zum SOVA Dekodierungsalgorithmus:

Zunächst werden für JEDEN STATE, zu JEDEM ZEITPUNKT die **Metriken** berechnet. Diese ergeben sich aus dem **Maximum** der beiden Metriken, die sich durch die (zwei) verschiedenen möglichen Zweige zu den jeweiligen Zuständen ergeben.  
Eine hohe Metrik bedeutet, dass mit hoher Wahrscheinlichkeit dieser State/Pfad korrekt ist, eine niedrige Metrik bedeutet, dass dieser State/Pfad eher unwahrscheinlich ist.  
Die Metrik im Zustand 0 zum Zeitpunkt 0 wird mit 0 initialisiert (da state 0 der vordefinierte Startzustand ist). Die Metriken im Zustand 0 zu allen anderen Zeitpunkten mit -1000 um diese als sehr unwahrscheinlich (praktisch unmöglich) zu markieren.  
Die Metrik errechnet sich aus der Summe von Metrik des previous states und dem Skalarprodukt aus erwarteter Codefolge und tatsächlich erhaltener Codefolge. (siehe IuK-Theorie S.227/228 Beispiel 8.6.7) Skalarprodukt da mit Soft-Werten gerechnet wird.  
Aufgrund der höheren Metrik wird das survivor bit, der previous\_matrix\_decision index gespeichert. Zusätzlich wird das **Delta** berechnet welches der Unterschied zwischen den Metriken des survivor Pfads und des verworfenen Pfads ist.

Nach der Metrikberechnung wird der survivor Pfad rekonstruiert, indem die survivor states gesucht werden.  
Jedoch ist zunächst wichtig ob das Signal terminiert wurde oder nicht. Falls es terminiert wurde, wird der letzte survivor state auf 0 gesetzt. (Ein terminiertes Signal endet immer im 0-Zustand). Falls es nicht terminiert wurde, wird zum letzten Zeitpunkt jener Zustand mit der höchsten Metrik gesucht und zum Endzustand.

**Survivor Pfad:** es wird im Trellis von hinten nach vorne (rechts nach links) gegangen. Mit dem Endzustand beginnend werden die survivor sates über die survivor bits ermittelt.  
Hier ist wieder die previous\_matrix\_decision Matrix wichtig. Kann alleine aus dem survivor bit nicht sicher gesagt werden was der vorherige state war (weil, wie eingangs erwähnt, etwa zwei Zustände mit demselben bit in den neuen Zustand wechseln) muss in die previous\_matrix\_decision Matrix geschaut werden, in welche der zwei previous Matrizen der richtige previous state steht.

Wichtig: survivor states des survivor Pfads dürfen nicht -1 werden.

Statt mit Wahrscheinlichkeiten zu rechnen wird die L-Algebra verwendet, die auf die Deltas angewendet wird:  
Beginnend zum Zeitpunkt wo erstmals 2 Pfade für einen Zustand möglich sind (in den Standardbeispielen bei t=3) passiert folgendes:  
Die Deltas werden folgendermaßen im survivor Pfad ge-updated: Delta wird mit bisherigem (im Trellis nach rechts gesehen) minimalen Delta **überschrieben** wenn sich das **survivor bit** des survivor states und des alternativen (verworfenen) states **unterscheiden**!

Berechnung Soft-Output:  
delta \* survivor\_bit – Zuverlässigkeitswerte(Eingangswslk.) – Lc(Channel Reliability) \* x\_d  
(siehe IuK-Theorie Formeln 8.75 und 8.78)

Zu delta:  
niedrig … ungewisse Entscheidung, Fehlerwahrscheinlichkeit hoch, survivor bit wird abgewertet

25/03/2016

Rcpp und Package Erstellung

Quelle: Advanced R  
@book{wickham2014advanced,

title={Advanced R},

author={Wickham, Hadley},

year={2014},

publisher={CRC Press}

}

<http://adv-r.had.co.nz/Rcpp.html>

(Codebeispiele sind auf der Website aufgeführt!!!)

C++ code wird in cpp Files geschrieben  
MÜSSEN mit folgenden Zeilen starten

#include <Rcpp.h>  
using namespace Rcpp;

Funktionen die in R verfügbar sein sollen müssen den Prefix  
// [[Rcpp::export]]  
haben.

Stellt nützliche Datentypen im C++ Code zur Verfügung:

|  |  |  |
| --- | --- | --- |
| Vektoren | Primitive | Matrix |
| NumericVector | double | NumericMatrix |
| IntegerVector | int | IntegerMatrix |
| CharacterVector | String | CharacterMatrix |
| LogicalVector | bool | LogicalMatrix |

R Code kann im cpp File mit speziellem Syntax hinzugefügt werden:  
/\*\*\* R  
# this is R code  
\*/

Sugar:

Es können auch Funktionen übergeben werden, die auf bspw. weiteren Argumenten angewendet werden! (Datentyp: Function)

Viele Operatoren sind auch, wie in R, auf einen ganzen Vektor anwendbar. Z.B. sqrt() angewendet auf einen NumericVector ergibt wiederum einen Vektor mit den Quadratwurzeln der ursprünglichen Elemente.  
Ausgewählte Operatoren: + \*, -, /, pow, <, <=, … (vollständige Liste der Quelle zu entnehmen)

STL (standard template library)

Stellt weitere nützliche Datenstrukturen und Funktionen/Algorithmen zur Verfügung

* Iteratoren
* Vectors
* Sets
* Maps
* …

Creating a Package  
(helpful YouTube video: https://www.youtube.com/watch?v=9PyQlbAEujY)

File -> New Project -> … -> R Package

Prerequisite:  
roxygen2 and devtools installed

Configure Build Tools  
-> use devtools (check)

R folder for R files  
man folder for documentation  
src folder for files to be compiled

To make function available when package is in use later, add roxygen annotation ‘@export’  
roxygen comments start with #’

To test package: Build tab -> Build & Reload (builds package and sets global environment)  
To share package: Build tab -> More -> Build Source Package (source code visible!), Build Binary Package (source code not visible)

Write cpp file -> put into src folder  
don’t forget include and namespace and // [[Rcpp::export]] for C++ functions accessibly from R. (As mentionen above)

Vorher: dyn.load um DLL zu laden  
Jetzt: Jede R Funktion, die eine C++ Funktion verwendet, bekommt die Annotation

#' @useDynLib channelcoding

#' @importFrom Rcpp sourceCpp

(see: <http://r-pkgs.had.co.nz/src.html#src>)