Tampereen ammattikorkeakoulu, amk-tutkinto
Tietotekniikan koulutusohjelma
Sulautetut järjestelmät
Patrick Saikkonen

Opinnäytetyö

VITERBI-ALGORITMIN KÄYTTÖ KONVOLUUTIOKOODATUN LÄHETTEEN PURUSSA VIRHEENHAVAINNOINTI- JA VIRHEENKORJAUSMENETELMÄNÄ

Työn ohjaaja: Yliopettaja Mauri Inha
Tampere 2009

Tampereen ammattikorkeakoulu, amk-tutkinto
Tietotekniikan koulutusohjelma, sulautetut järjestelmät

Tekijä: Patrick Saikkonen
Työn nimi: Viterbi
Päivämäärä: 30.4.2009
Työn laajuus: 42 sivua
Avainsanat: Viterbi, konvoluutiokoodaus
Koulutusohjelma: Tietotekniikka
Suuntautuminen: Sulautetut järjestelmät
Työn ohjaaja: Yliopettaja Mauri Inha

Tiivistelmä

Tässä opinnäytetyössä aiheena on konvoluutiokoodauksen käyttö virheenhavainnointi- ja virheenkorjausmenetelmänä ja konvoluutiokoodatun lähetteen purkualgoritmien toimintaan tutustuminen.

Digitaalista tiedonsiirtoa käytetään nykyään yhä useammissa sovelluksissa. Digitaalinen tiedonsiirto asettaa uusia haasteita lähetteen perille saamiseen alkuperäisessä muodossa. Konvoluutio on yksi tärkeimmistä digitaalisen signaalinkäsittelyn seikoista. Konvoluutio saadaan aikaiseksi suorittamalla summaus lähetettävien bittien ja koodaajan siirtorekisterissä olevien bittien kesken. Tätä tapahtumaa kutsutaan konvoluutiokoodaukseksi.

Konvoluutiokoodauksella pyritään minimoimaan digitaalisessa tiedonsiirrossa ilmestyvät virheet. Kanavalla oleva kohina voi aiheuttaa lähetteen vääristymistä, kun bitti vaihtuu matkalla toiseksi. Konvoluutiokoodauksen avulla virheen lähetteeseen aiheuttama haitta saadaan pienemmäksi, koska lähetyksessä käytetään useampia bittejä kuvastamaan yhtä tilan vaihtumista.

Konvoluutiokoodaus itsessään ei kuitenkaan vielä ratkaise ongelmia. Tämän lisäksi purkupuolella tarvitaan algoritmi, joka purkaa vastaanotetun konvoluutiokoodatun lähetteen alkuperäiseksi viestiksi. Tähän käyttöön on kehitetty useita toistensa kaltaisia algoritmeja, joista tässä työssä käydään tarkemmin läpi Viterbi-algoritmi. Viterbi-algoritmin toiminnan ymmärtämiseksi työssä käydään myös Markovin mallien periaatteet.

Viterbi-algoritmi on dynaaminen algoritmi. Algoritmin toimintatapaan tutustutaan muutamalla havainnollisella ja yksinkertaisella esimerkillä. Esimerkkien avulla on helppo ymmärtää konvoluutiokoodauksen käyttö. Esimerkit näyttävät myös miten konvoluutiokoodattu lähete puretaan ja miten lähetyksessä olevat virheet havaitaan.

Avainsanat Viterbi-algoritmi, konvoluutiokoodaus, Markovin piilomalli

TAMK University of Applied Sciences, Bachelor's degree
Information Technology, Embedded Systems

Author: Patrick Saikkonen
Work label: Viterbi
Date: 30.4.2009
Number of pages: 42
Keywords: Viterbi, convolutional coding
Education program: Information Technology
Line: Embedded Systems
Thesis supervisor: Senior Lecturer Mauri Inha

Abstract

This thesis concentrates on convolutional coding used as a system to detect and correct errors when sending digital information. The main principles of encoding algorithms are also explained.

Transferring data in digital form is used more and more in different kind of applications. The main idea of convolutional coding is to minimize errors in digital data transfer. The noise in the channel can cause trouble when receiving the message, if one or more of the bits has turned into another. Convolutional coding can be used to solve this problem, because the main principle in it is to send multiple bits when state changes.

Convolutional coding doesn't solve the problem on its own. An algorithm to encode the received message is also needed. There are many algorithms invented for the particular use. This thesis concentrates on Viterbi-algorithm that is widely used in information technology to encode convolutional coded messages.

Viterbi-algorithm is a dynamic algorithm. The basics of Viterbi-algorithm are shown with few simple examples that are explained step by step. Examples also show how convolutional coding is used and how Viterbi-algorithm can get rid of most of the errors.

Keywords Viterbi-algorithm, convolutional coding, Hidden Markov model

Esipuhe

Tämä työ on tehty Tampereen ammattikorkeakoulun sulautettujen järjestelmien opintolinjan insinöörityönä.

Kiitokset kuuluvat kaikille Tampereen ammattikorkeakoulun opettajille, jotka ovat jakaneet tietojaan ja taitojaan. Erityiskiitokset kuuluvat aiheen esille tuoneelle koulutuspäällikkö Ari Rantalalle, työtä ohjanneelle yliopettaja Mauri Inhalle ja kieliasun oikeellisuutta ylläpitäneelle lehtori Esa Kylänpäälle. Lopuksi haluan vielä kiittää oikoluvusta ja tekstin yleisilmeen tarkistuksesta Maarit Huikkua ja Kati Rintalaa.

Tampereella 30.4.2009

Patrick Saikkonen

Sisällysluettelo

1 Johdanto

Digitaalista tiedonsiirtoa käytetään nykyään yhä enenevässä määrin. Digitaalinen tiedonsiirto on muun muassa mahdollistanut langattomat puheyhteydet. Langattomassa yhteydessä tulee vastaan kuitenkin uusia haittatekijöitä, jotka häiritsevät lähetyksen pääsemistä perille alkuperäisessä muodossa. Digitaalisessa muodossa lähetettävien viestien ongelmaksi muodostuu helposti kohinainen kanava.

Kohinaisella kanavalla bitti voi kääntyä, jolloin vastaanotettu viesti on erilainen kuin alun perin lähetetty viesti. Tämän ongelman minimoimiseksi käytetään konvoluutiokoodausta. Konvoluutiokoodauksen avulla saadaan muodostettua yhdestä lähetettävästä bitistä useampi bitti. Tällaista menetelmää käyttämällä yhden bitin virhe on suhteessa pienempi ja näin ollen helpommin korjattavissa.

Konvoluutiokoodauksessa lähetettävät bitit riippuvat järjestelmän sen hetkisestä tilasta ja koodaajalle sisään annettavasta bitistä. Muodostuvat bitit ovat kanavasymbolibittejä, joiden purkamiseen käytetään siihen erikoistunutta algoritmia. Konvoluutiokoodatun lähetteen purussa on käytössä monia eri algoritmeja, ja eräs näistä on tässä työssä tarkemmin käsitelty Viterbi-algoritmi.

Viterbi-algoritmi on laajasti käytössä konvoluutiokoodattujen lähetteiden purussa ja on yksi tärkeimpiä algoritmeja digitaalisessa signaalinkäsittelyssä /2, III/. Tämän kirjallisen esityksen tavoitteena on antaa muutaman esimerkin avulla kuva siitä, miten algoritmi varsinaisesti toimii. Algoritmin periaate perustuu todennäköisyyksien laskentaan. Viterbi-algoritmi laskee vastaanotettujen bittien perusteella todennäköisintä sekvenssiä käyttämällä laskennassa yksinkertaista Hamming-etäisyyttä. Tällaisen laskennan avulla voidaan päätellä onko vastaanotettuihin bitteihin tullut virhe matkalla. Lopullisen puretun viestin eli niin sanotun Viterbi-reitin pituus on sama kuin virheiden lukumäärä.

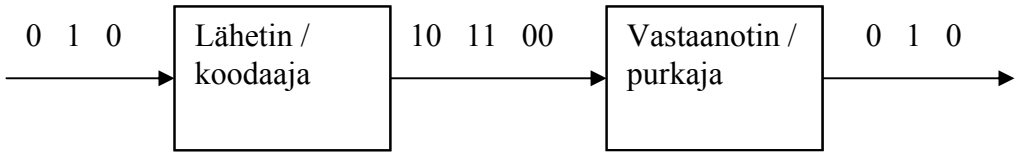
Algoritmi ei takaa kaikkien virheiden löytymistä, mutta pitkällä tähtäimellä suurin osa konvoluutiokoodatussa lähetteessä olevista virheistä löydetään. Virheitten löydyttyä on helppo palauttaa alun perin koodattu viesti purkupuolella. Jos kuitenkin jokin virheistä jää korjaamatta, tulee puretusta viestistä erilainen kuin lähetetystä viestistä. Virheiden havainnointia voidaan parantaa käyttämällä suurempaa purkuikkunaa Viterbi-algoritmin apuna. Purkuikkunan iso koko auttaa algoritmia saamaan oikean reitin varmemmin selville. Purkuikkunaa suurentaessa kasvaa kuitenkin myös muistin tarve. Tämä johtuu siitä, että purkuikkunan kohdalla Viterbi-algoritmi säilyttää muistissa useita sekvenssejä, jotta saataisiin selville mikä näistä on todennäköisin. Vähemmän todennäköiset reitit poistetaan hyvissä ajoin, mutta joissain tapauksissa voidaan myös niistä pitää kirjaa.

Algoritmiin kohdistuvat vaatimukset kasvavat jatkuvasti. Kehitteillä on aina uudenlaista sovellusmahdollisuutta ja suuremmalla toteutusnopeudella. Uusia lähestymistapoja koodauksen purkamiseksi nopeammin ja tehokkaammin on jatkuvasti kehitteillä. /2, III/

2 Konvoluutiokoodaus

Konvoluutio on matemaattinen tapa yhdistää kaksi signaalia, jotta saadaan muodostettua kolmas signaali. Se on tärkein yksittäinen asia digitaalisessa signaalinkäsittelyssä. Tietoliikennetekniikassa konvoluutiokoodausta käytetään virheenhavainnointi- ja virheenkorjausmenetelmänä.

Konvoluutiokoodaaja tuottaa yhtä lähetettävää bittiä kohden useamman bitin. Lähetettävät bitit riippuvat tulevasta bitistä ja siitä, missä tilassa koodaaja kulloinkin on. Koodaajan tilaan puolestaan vaikuttavat aiemmin tulleet bitit. Konvoluutiokoodaaja sisältää siirtorekisterin. Tuleva bitti koodataan rekisterissä olevien bittien kanssa käyttäen modulo-2-yhteenlaskua ja tämä yhteenlaskun tuloksesta muodostuu konvoluutiokoodattu lähete. Kuvio 1 näyttää konvoluutiokoodauksen periaatekuvan.



Kuvio 1: Konvoluutiokoodauksen perusperiaate

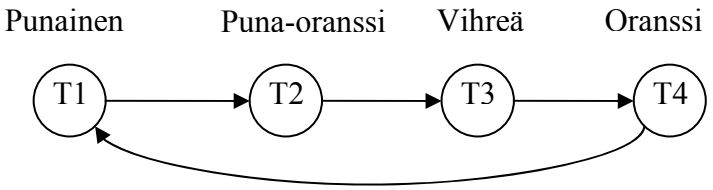
Konvoluutiokoodaus on yksinkertainen operaatio, joka ei vaadi suurta muistia. Konvoluutiokoodatun lähetteen purkaminen puolestaan vaatii vastaanottopuolelta muistikapasiteettia ja tehoa. Konvoluutiokoodauksen purussa eli dekoodauksessa käytetään niin sanottua Viterbi-algoritmia. Konvoluutiokoodausta voidaankin hyvin käyttää esimerkiksi lähetettäessä satelliitista tietoa maan pinnalle. Koodauksessa tarvittava energia on pieni, eikä vastaanottopuolella ole kovin tiukkoja rajoituksia energian suhteen.

Konvoluutiokoodausta käytettäessä ei voida olla varmoja, että jokainen yhden bitin virhe tulee korjattua, mutta konvoluutiokoodaus on hyvä keino saada tietty osuus bittivirheitä korjattua pitkällä aikavälillä.

3 Johdatus Markovin mallin käyttöön sekvenssejä sisältävissä järjestelmissä

Usein erilaisissa tekniikan järjestelmissä joudutaan tekemisiin jonkinlaisten sekvenssien tai kaavojen kanssa. Konvoluutiokoodauksessa sekvenssi muodostetaan koodaajan sisään tulevien bittien ja siirtorekisterin avulla. Sekvenssi tarkoittaa tällöin lähetettä. Koodaajan tilasekvenssi ja koodaajasta ulostuleva kanavasymbolisekvenssi noudattavat tiettyä järjestystä, joka tiedetään myös koodinpurkupuolella. Kaikki järjestelmät eivät kuitenkaan ole samanlaisia.

Yksinkertainen sekvenssi ei välttämättä vaadi kuin tilasta tilaan siirtymistä edellisen tilan perusteella. Tällaista järjestelmää on helppo ymmärtää ja analysoida. Esimerkki tällaisesta järjestelmästä on liikennevalot. Liikennevaloissa sekvenssi on aina sama ja tietty tila johtaa aina samaan seuraavaan tilaan kuin aiemmilla läpikäynneillä. Liikennevalojen tilakone voidaan nähdä kuviossa 2. /4/

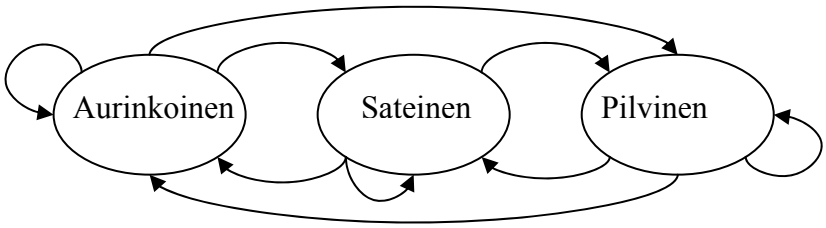


Kuvio 2: Liikennevalojen tilakone

Liikennevalojen sekvenssiä tarkastellessa tiedetään aina missä tilassa tilakone kulloinkin on. Liikennevalosekvenssiä monimutkaisemmissa sekvensseissä ei nykyisen tai seuraavan tilan määritys ole enää niin suoraviivaista. Monimutkaisemman järjestelmän esimerkiksi sopii säätila. Kuvitellaan sää kolmena eri tilana: aurinkoinen, pilvinen ja sateinen. Näiden ei voida olettaa seuraavan toisiaan tietyssä järjestyksessä. Tällainen esimerkki on normaali Markovin malli. Esimerkissä voidaan käyttää Markovin oletusta, jolla säätilan ennustamista saadaan huomattavasti helpotettua.

Markovin oletuksen mukaan seuraava tila riippuu vain aiemmista tiloista. Tarkasteluun voitaisiin ottaa muutaman aiemman päivän säätila ja tämän perusteella ennustaa tuleva sää. Tällaisessa järjestelmässä ainoa parametri on tilojen välinen muutostodennäköisyys. Varsinaiseen säätilan ennustamiseen tällainen ei kuitenkaan sovellu, koska monia tekijöitä jäisi huomioimatta yksinkertaistuksen vuoksi. /4/

Kuvio 3 näyttää kaikki mahdolliset kolmen eri säätilan muutokset.



Kuvio 3: Esimerkin säätilojen tilakone

Normaalissa Markovin mallissa ainut parametri on muutostodennäköisyys tilojen välillä. Tämä tarkoittaa todennäköisyyttä sille, kuinka mahdollinen tietty tilasiirtymä tilojen välillä on. Markovin malli olettaa muutostodennäköisyyksien pysyvän samoina ajan kuluessa. Täten kyseisistä todennäköisyyksistä voidaan luoda taulukko. Oletus on tärkeä osa Markovin mallia, vaikkakaan se ei usein ole kovin realistinen. /4/

Tilamuutostaulukko esimerkin säätilan ennustuksia varten voidaan nähdä taulukosta 1, jossa todennäköisyysprosentti on esitetty desimaaliluvulla nollasta yhteen. Taulukosta nähdään, että aurinkoisen päivän jälkeen seuraava päivä on aurinkoinen 35 % todennäköisyydellä ja 25 % todennäköisyydellä sataa. Todennäköisintä seuraavana päivänä on aurinkoisen päivän jälkeen pilvinen sää.

Taulukko 1: Esimerkki muutostodennäköisyyksistä Markovin mallissa

		Sää tänään		
		Aurinkoinen	Pilvinen	Sateinen
Sää eilen	Aurinkoinen	0,35	0,40	0,25
	Pilvinen	0,35	0,30	0,35
	Sateinen	0,30	0,30	0,40

3.1 Markovin prosessi

Markovin prosessi on matemaattinen malli muistittomalle järjestelmälle, jossa tutkitaan järjestelmän kehittymistä tai sen luomaa sekvenssiä. Markovin prosessi liikkuu tilasta tilaan vain aiempien tilojen perusteella. Se kuinka moni aiemmista tiloista otetaan huomioon siirryttäessä seuraavaan tilaan, riippuu Markovin prosessin asteluvusta. Yksinkertaisimmassa eli ensimmäisen asteluvun Markovin prosessissa käytetään vain nykyistä tilaa, jotta saadaan lasketuksi todennäköisin seuraava tila. Malli ei huomioi tässä vaiheessa enää aiempia tiloja, joita on kuitenkin käytetty nykyisen tilan laskemiseen. Aiemmin käsitelty säätilaesimerkki oli normaalin Markovin mallin mukainen.

3.2 Markovin piilomalli

Kaikissa järjestelmissä, joita halutaan mallintaa, ei voida käyttää Markovin prosessia tehokkaasti. Joissakin tapauksissa ei voida suoraan nähdä, missä tilassa mallinnettava järjestelmä on. Tällöin käytetään järjestelmän tulkitsemisessa muita parametreja, niin sanottuja tarkkailtavia tiloja, joiden avulla voidaan suurella todennäköisyydellä päätellä, missä tilassa järjestelmä kulloinkin on. Kun järjestelmän mallintamisessa tarvitaan tarkkailtavia tiloja, on kyseessä Markovin piilomalli.

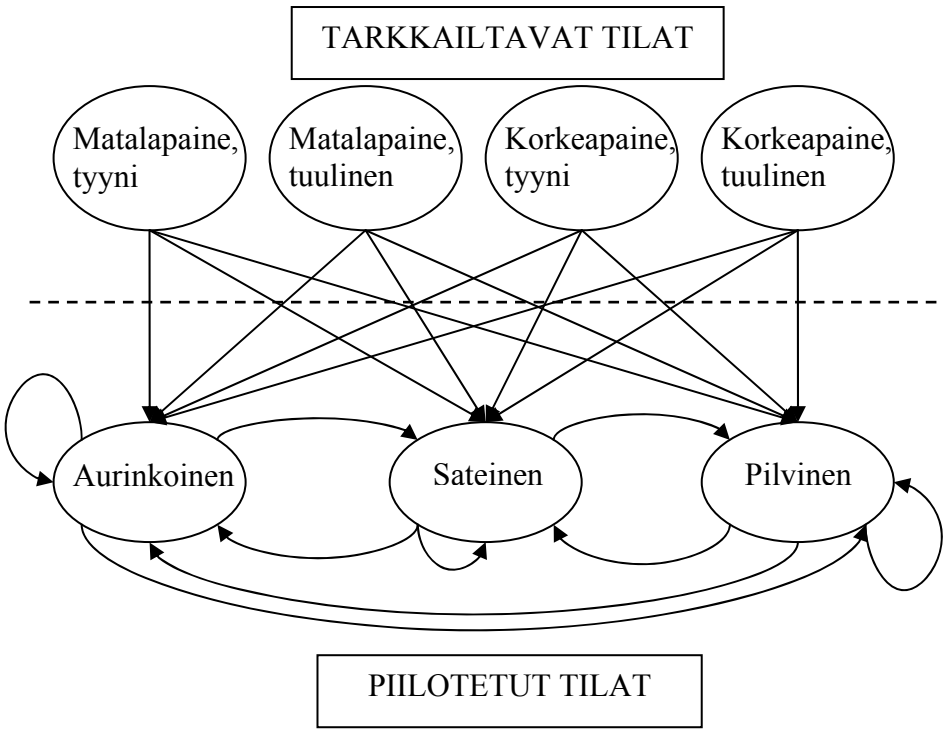
Markovin piilomalli (englanniksi Hidden Markov Model, HMM) on tilastollinen malli, johon myös Viterbi-algoritmin toiminta perustuu. Koodinpurussa käytetty Markovin piilomalli olettaa mallinnettavan järjestelmän olevan Markovin prosessi tuntemattomilla parametreilla. Tarkoitus on selvittää tarkkailtavan datan avulla piilotetut parametrit. Ulossaatavia parametreja voidaan sitten käyttää tarkemmassa analyysissä, mikä tapahtuu erilaisilla kaavantunnistusohjelmilla.

Hyvä esimerkki piilotetun Markovin mallin todellisesta käytöstä on puheentunnistus. Kuultu ääni riippuu kurkun suuruudesta, kielen asennosta ja monista muista seikoista. Puheentunnistuksessa tulee ottaa huomioon äänen muutokset hyvinkin tarkasti, jotta tiedetään oikea sekvenssi sanoja, joita juuri tunnistettiin. Jotkin puheentunnistajat toimivat olettamalla tulevan äänen piilotettuina tiloina, jolloin äänen muutoksia seuraamalla voidaan päätellä niistä koostuvat sanat.

3.3 Markovin piilomallin rakenne

Markovin piilomalli poikkeaa normaalista Markovin mallista oleellisesti siten, että tila ei ole suoraan näkyvissä tarkkailijalle. Normaalissa Markovin mallissa tilojen muutostodennäköisyydet ovat ainoat parametrit. Markovin piilomalli käsittelee tilannetta, jossa tila ei suoraan ole näkyvissä. Tietyn tilan toteutuminen voidaan kuitenkin päätellä tarkkailtavista tiloista, koska jokaisella tilalla todennäköisyydet ovat jakautuneet tiettyihin ulostuloihin. Markovin piilomallin luoma merkkisekvenssi antaa tietoa tilojen sekvenssistä.

Kuvio 4 on esimerkki Markovin piilomallin rakenteesta.



Kuvio 4: Markovin piilomallin rakenne

Aiemmin esitettyyn säätilaesimerkkiin on nyt otettu mukaan tarkkailtavat tilat kuviossa 4. Tarkkailtavat tilat voivat tulla esimerkiksi erilaisilta antureilta, eikä tällöin tarvitse varsinaisesti paikan päällä käydä säätä tarkkailemassa. Esimerkin tarkkailtaviksi tiloiksi on otettu ilmanpaine ja tuuli ja niistä syntyvät yhdistelmät. Jokainen tarkkailtava tila pitää sisällään todennäköisyyden siitä, kuinka todennäköisesti jokin piilotettu tila on sillä hetkellä voimassa. Piilotettujen tilojen ajatellaan käyttäytyvän ensimmäisen asteen Markovin prosessin mukaisesti, joten ne ovat kaikki yhteydessä toisiinsa.

Tarkkailtavien tilojen ja piilotettujen tilojen välisistä todennäköisyyksistä luodaan Markovin piilomallissa myös taulukko. Esimerkki tällaisesta on muodostettu taulukkoon 2.

Taulukko 2: Esimerkki tarkkailtavien tilojen ja piilotettujen tilojen välisistä todennäköisyyksistä

	Matalapaine, tyyni	Matalapaine, tuulinen	Korkeapaine, tyyni	Korkeapaine, tuulinen
Aurinkoinen	0,10	0,05	0,50	0,35
Pilvinen	0,30	0,30	0,10	0,30
Sateinen	0,20	0,40	0,10	0,30

Taulukon 2 mukaisia todennäköisyyksiä käytetään yhdessä piilotettujen tilojen esiintymistodennäköisyyksien kanssa. Laskentaan tulee sisällyttää molemmat, jotta voidaan tietää, mihin tilaan järjestelmä kulloinkin päätyy. Taulukon jokaisen vaakarivin summan tulee olla yhteensä 1.

4 Viterbi-algoritmi

Viterbi-algoritmi on dynaaminen algoritmi, jota käytetään konvoluutiokoodattujen lähetteiden purkuun. Viterbi-algoritmi on laajasti käytetty kaikenlaisessa digitaalisessa kommunikaatiossa sen tehokkuutensa ansiosta. Se on tehokas tapa purkaa konvoluutiokoodattuja lähetteitä, vaikka kanavalla olisi suurtakin kohinaa joukossa.

Algoritmin perusperiaatteen mukaan se tutkii todennäköisintä tilasekvenssiä, joka johti tiettyihin ulostuloihin tai parametreihin. Algoritmi pitäytyy aina todennäköisimmäksi katsomassaan sekvenssissä tai sekvensseissä. Todennäköisimpiä sekvenssejä voi olla samaan aikaan useita, mutta lopussa vain yksi voi tulla valituksi todennäköisimmäksi sekvenssiksi. Tämä sekvenssi, jota myös kutsutaan nimellä Viterbi-reitti (englanniksi Viterbi-path), tulee ulos koodinpurkajasta ja näin koodattu lähete on saatu purettua ja mahdolliset virheet on korjattu.

4.1 Historia

Viterbi-algoritmin kehitti ja keksi amerikanitalialainen insinööri Andrew J. Viterbi, joka oli myöhemmin myös Qualcom Corporationin (1985) perustajajäsen /3, s. 4/. Hänen tarkoituksensa oli toteuttaa algoritmi, joka toimisi virheenkorjaavana järjestelmänä digitaalisille kommunikaatiolinkeille. Andrew J. Viterbin materiaalit liittyen Viterbi-algoritmiin julkaistiin vuonna 1967. Hänen ajatuksensa oli käyttää Viterbi-algoritmia konvoluutiokoodattujen lähetteiden purkuun, ja tässä tarkoituksessa algoritmi on yhä monissa sovelluksissa.

Algoritmin julkistamisen jälkeen monet muut tutkijat ovat laajentaneet Viterbin työtä muun muassa kehittämällä Viterbi-algoritmiin yhteensopivia konvoluutiokoodeja, etsimällä suorituskyvyn rajoja algoritmille ja muutellen koodinpurkajan suunnitteluparametreja optimoidakseen tekniikan käytön sekä laitepuolella että ohjelmapuolella. /3, s. 4/

Viterbi-koodinpurkualgoritmia on myös käytetty purkamaan trelliskoodattua modulaatiota. Trelliskoodausta käytetään puhelinlinjamodeemeissa. Trelliskoodaus tarkoittaa, että koodauksessa käytetään tietynlaista kaaviota, jonka mukaan tietystä tilasta siirrytään toiseen tilaan riippuen siitä mikä koodaajalle tulee sisäänmenoksi.

Kyseistä kaaviota kutsutaan Trellis-kaavioksi. Trellis-kaavion avulla koodauksesta käydään esimerkki tämän työn kuluessa.

Algoritmin kehittäjä Andrew J. Viterbi oli ehdolla Millenium-teknologiapalkinnon saajaksi kehittämästään Viterbi-algoritmista vuonna 2008. Miljoonan dollarin Millenium-palkinto annettiin tuolloin kuitenkin Robert Langerille. /7/

4.2 Tarkempi katsaus Viterbi-algoritmin toimintaan

Viterbi-koodinpurkaja operoi koodia kehys kerrallaan samanlaisen Trellis-kaavion läpi, jota koodauksessa on käytetty. Tarkoitus on selvittää koodauksen toimintamalli. Viterbi-algoritmi käsittää tulevan koodin rajallisena määränä tiloja, jotka voidaan listata. Nämä tilat käsitetään solmuina (eng. node). Tietyllä ajanjaksolla t solmuja on tietyn lukumäärän verran. Siirryttäessä seuraavaan ajanjaksoon t+1, tulee Viterbi-algoritmin selvittää, todennäköisyyksiä laskien, mitä reittiä koodattu lähete noudattaa. Vain yksi reitti voi lopulta tulla valituksi vaikka yhtä todennäköisiä reittejä olisi useita. Tätä reittiä kutsutaan selviytyjäreitiksi. /2, s. 27/

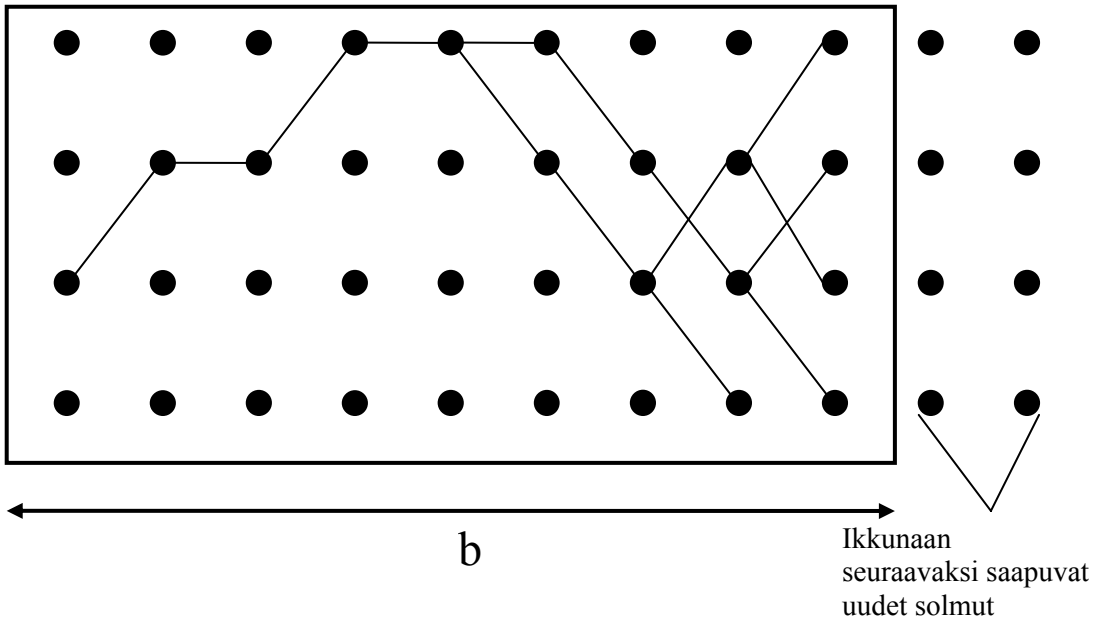
Koska vain yksi reitti voi tulla valituksi, tulee mahdollisista muista reiteistä pitää kirjaa niin kauan, kunnes ne todetaan epätodennäköisiksi. Joissakin sovelluksissa on mahdollista pitää kirjaa hyvinkin pitkältä ajalta, mutta tämä vaatii suurta muistitilaa. Algoritmin suorituskyky ja tilavaatimukset paranevat, kun päätös todennäköisimmästä reitistä tehdään mahdollisimman nopeasti. Joissakin tapauksissa historia eri reiteistä voi olla täydellinen: koodatun lähetteen aloitustila on tiedossa ja tilaa on tarpeeksi pitää muistissa kaikki mahdolliset reitit. Näin ei kuitenkaan usein ole, vaan täytyy löytää ohjelmallinen ratkaisu, jotta suorituskyky säilyisi.

Viterbi-algoritmi selvittää selviytyvän reitin todennäköisyyden perusteella. Tämä todennäköisyys saadaan aikaan ottamalla huomioon tulevien tilojen todennäköisyydet, edellisten tilojen vaikutukset ja muut tiedossa olevat parametrit.

4.3 Viterbi-algoritmin käyttämä purkuikkuna

Ennen kuin Viterbi-algoritmin toimintaperiaatetta alkaa tarkastella, on hyvä tietää, mikä on Trellis-kaavio ja miten Viterbi-algoritmi sitä käsittelee. Trellis-kaavio on kaavio, jossa solmut ovat järjestyksessä pystysuuntaisesti. Tämä pystysuuntainen viipale Trellis-kaaviossa on yksi ajanhetki. Jokainen solmu jokaisella ajanhetkellä yhdistyy vähintään yhteen solmuun aiemmasta ajanhetkestä ja vähintään yhteen solmuun tulevasta ajanhetkestä. Trellisin alku- ja loppuajanhetkillä on solmuja vain yksi. Trellisiä käytetään koodaajissa ja koodinpurkajissa. /2, s. 27/

Kun algoritmi etenee Trellis-kaaviossa, on sillä apunaan niin sanottu purkuikkuna. Tämän pituus määrää, kuinka pitkältä ajalta reittiä pidetään tallessa. Ikkunassa ovat kaikki mahdolliset tilat tietyllä hetkellä kuten Trellis-kaaviossa. Ikkuna ei kuitenkaan käsitä koko Trellisiä, vaan ikkunan pituus määritellään sopivaksi. Tällainen ikkuna voidaan nähdä kuviossa 5.



Kuvio 5: Koodinpurkajan käyttämä purkuikkuna.

Purkuikkunassa (kuvio 5) solmut etenevät oikealta vasemmalle ajan kuluessa. Uudet solmut tulevat ruutuun jokaisella uudella ajanhetkellä ja vanhat solmut poistuvat vasemmalta ulos purkuikkunasta.

Purkuikkunan pituudeksi määriteltävän arvon, joka on kuviossa 5 muuttuja b, tulee olla riittävän suuri. Tämä arvo on useita kertoja suurempi kuin solmujen etäisyys toisistaan. Jos muuttuja b valitaan tarpeeksi suureksi, on hyvin todennäköistä, että ikkunan aikana saadaan todennäköisin reitti valituksi. Tätä auttaa myös se, että koodi on hyvin suunniteltu kanavalle. /2, s. 26/

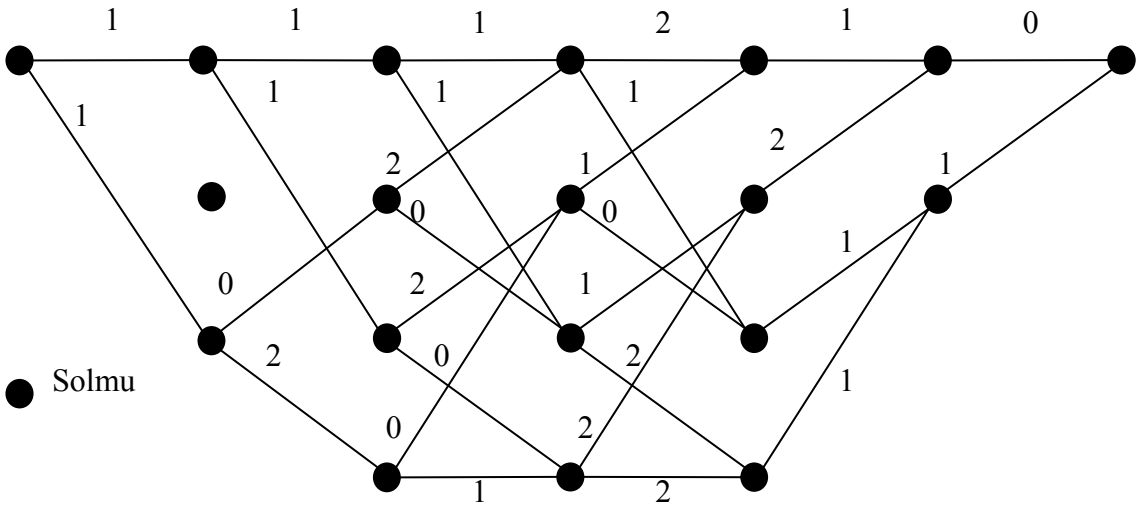
Kaikkein todennäköisin reitti saadaan selville, kun lisätään uuden reitin numero jo kasassa olevaan reitin pituuteen. Jos on lukumäärän x verran reittejä solmusta eteenpäin ja y on tähänastisen reitin kokonaispituus, on silloin olemassa x^y verran mahdollisia reittejä jokaiselle solmulle. Reitti, jolla on pienin numero, on todennäköisin reitti uuteen solmuun. /2, s. 26/

Kuten kuviosta 5 nähdään, Viterbi-reittiä on löydetty puoleen väliin asti ikkunassa, mutta loppupuoli on vielä varsin avoin. Todennäköisin reitti tulee kuitenkin valita ennen kuin on mahdollista antaa vanhojen solmujen kokonaan kadota ikkunasta. Tämä tarkoittaa siis sitä, että vasemmalta ikkunaan voi tulla vain yksi reitti, ei yhtään enempää. Puolestaan oikealta puolelta tuleviin solmuihin jatkuu jokaiseen kaksi uutta reittiä. Toinen näistä poistetaan välittömästi epätodennäköisempänä reittinä.

4.4 Viterbi-algoritmin toimintaperiaate

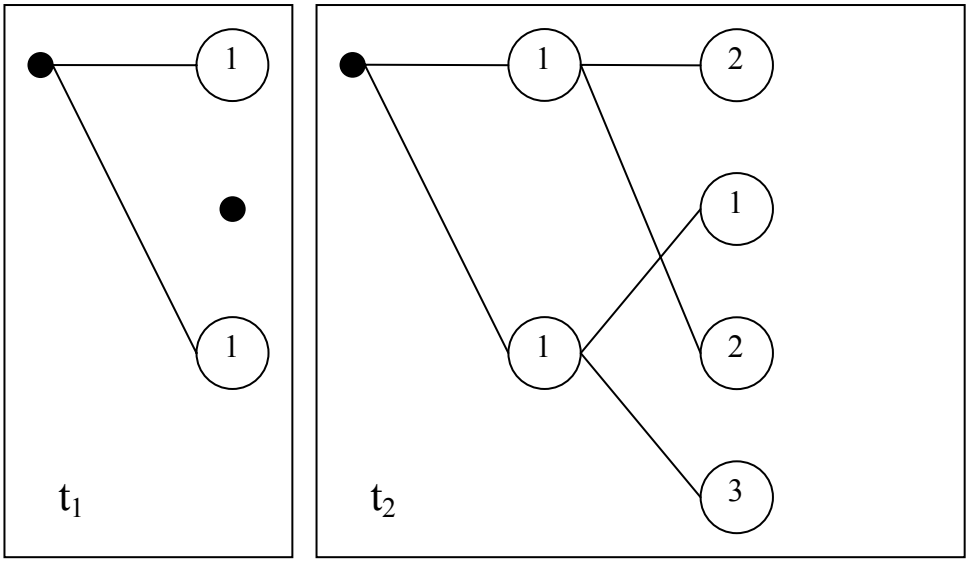
Viterbi-algoritmin toiminta on helpoin käsittää tila kerrallaan etenevänä suoraviivaisena laskentana. Seuraavaksi selvitetään Viterbin todellista käyttäytymistä neljätilaisen Trellis-kaavion avulla. Tiloina Trellis-kaaviossa voivat olla esimerkiksi 00, 01, 10 ja 11. /2, s. 28/

Algoritmi aloittaa jostakin tietystä pisteestä, kuten kuviossa 6. Solmujen väliset etäisyydet on merkitty solmuja yhdistävän viivan viereen. Etäisyys tunnetaan nimellä ero tai reitti (eng. metric, path tai discrepancy). Solmujen etäisyydet toisistaan tarkoittavat tarkemmin sanottuna todennäköisyyksiä tai painotuksia. Tämä lukuarvo (eng. branch metric) on todennäköisyys siitä, kuinka mahdollinen siirtymä on tiettyjen tilojen välillä. Lukuarvoista pidetään kirjaa ja reitin pituus saadaan selville lisäämällä uuden reitin pituus jo olemassa olevan reitin pituuteen. Tämä toistetaan jokaisessa uudessa kehyksessä.



Kuvio 6: Trellis-kaavio, johon on sijoitettu solmujen etäisyydet.

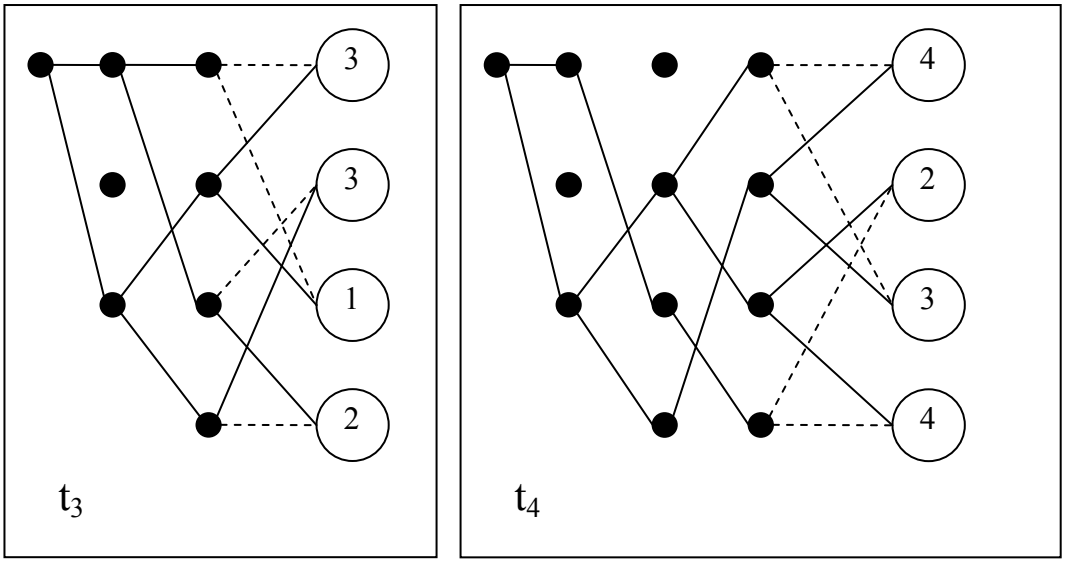
Seuraavista kuvista voi nähdä, miten Viterbi-algoritmi löytää todennäköisimmän reitin aloitussolmusta viimeiseen solmuun kuvion 6 mukaisessa tapauksessa. Kuvissa pallojen sisällä olevat numerot merkitsevät reitin pituutta siihen pisteeseen mennessä.



Kuvio 7: Koodinpurun eteneminen kahden ensimmäisen tilan osalta.

Kuviossa 7 ajanhetkellä t1 saadaan sisään ensimmäiset solmut. Tämä on ensimmäinen tila, jota Viterbi-algoritmi käsittelee. Reittivaihtoehtoja on vain kaksi, ja reitit ovat toisiinsa nähden samanpituisia tarkoittaen, että kumpi tahansa reiteistä on tässä vaiheessa yhtä todennäköinen. Nämä reitit tulee molemmat pitää tallessa, koska jatkosta ei vielä ole tietoa. Vaihtoehtoisesti Viterbi-algoritmi voisi tehdä valintansa jo tässä vaiheessa, jolloin muistitilaa reittien ylläpitämiseksi tarvittaisiin huomattavasti vähemmän. Tämä kuitenkin johtaisi myös siihen, ettei välttämättä löydettäisi kaikkein todennäköisintä reittiä.

Ajanhetkellä t2 saadaan Trellis-kaaviolle jatkoa neljästä uudesta solmusta. Vaihtoehtoisien reittien lukumäärä kaksinkertaistuu tässä kohtaa. Yhdestäkään reitistä ei vielä jouduta luopumaan, koska jokaiseen uuteen tilaan johtaa vain yksi reitti. Yksi reiteistä on tässä vaiheessa muita lyhyempi.

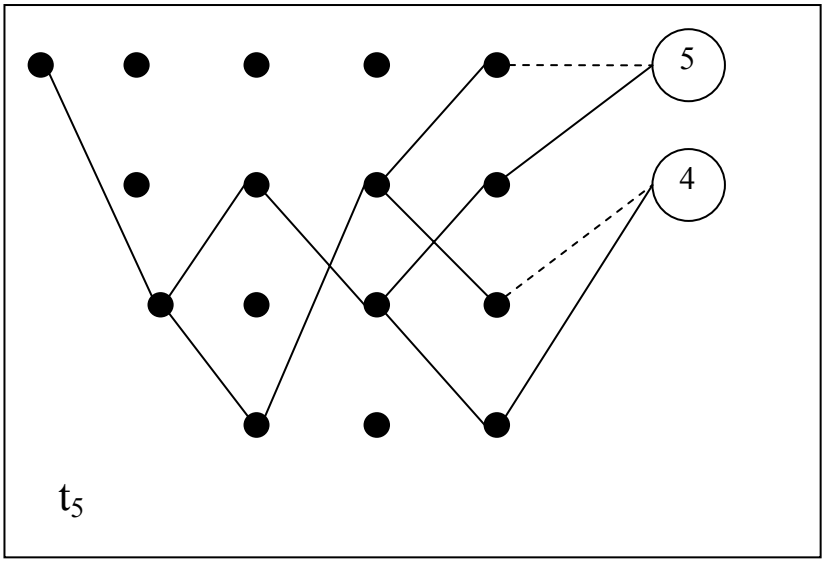


Kuvio 8: Koodinpurun eteneminen Viterbi-algoritmilla ajanhetkillä t3 ja t4.

Ajanhetkellä t3 eri reittien pituuksissa on syntynyt jo huomattavia eroja. Kuitenkin reittihistoria säilytetään. Reiteistä poistetaan vain ne, jotka ovat jo samanpituisia tai pidempiä verrattuna lyhyimpään reittiin tietyssä solmussa. Reittien poistossa voidaan käyttää tiettyä metodia, kuten ylemmän viivan poistamista, mutta tässä tapauksessa reiteistä on poistettu satunnaisesti toinen. Jokaiseen solmuun ajanhetkillä t3, t4 ja t5 tulee kaksi eri reittiä.

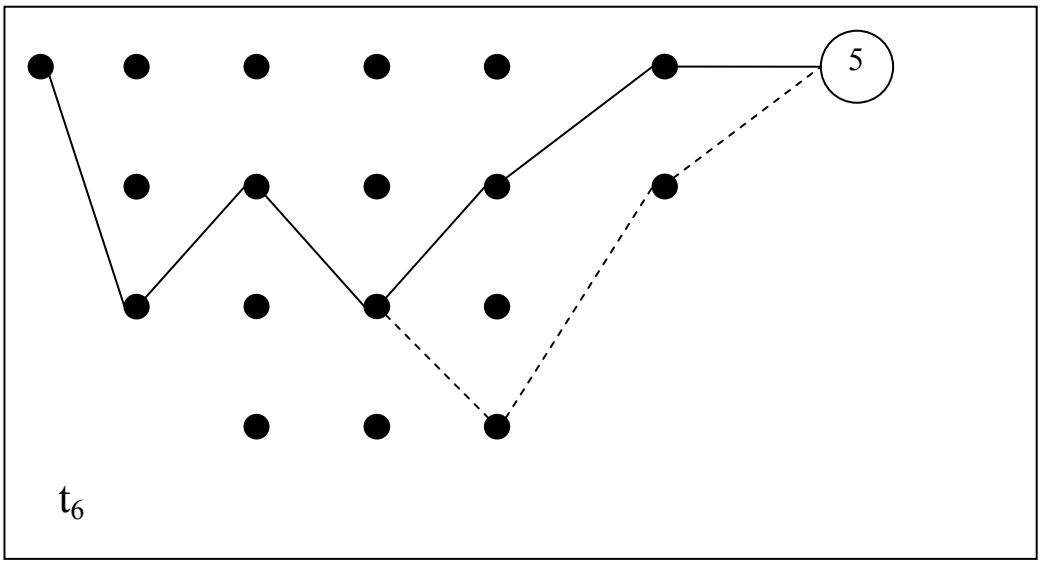
Ajanhetkestä t3 tultaessa ajanhetkeen t4 on katkenneiden reittien alkupäät poistettu. Myös ajanhetkellä t4 katkeaa useita reittejä. Solmuissa, joihin katkenneet reitit voisivat jatkua, on myös jokin muu lyhyempi reitti.

Reittejä on yhä useita mahdollisia mukana, mutta seuraavilla ajanhetkillä (t5 ja t6, kuvioissa 9 ja 10) joudutaan tekemään valintoja todennäköisimmän reitin löytämiseksi. Vain yksi reiteistä voi tulla lopulta valituksi.

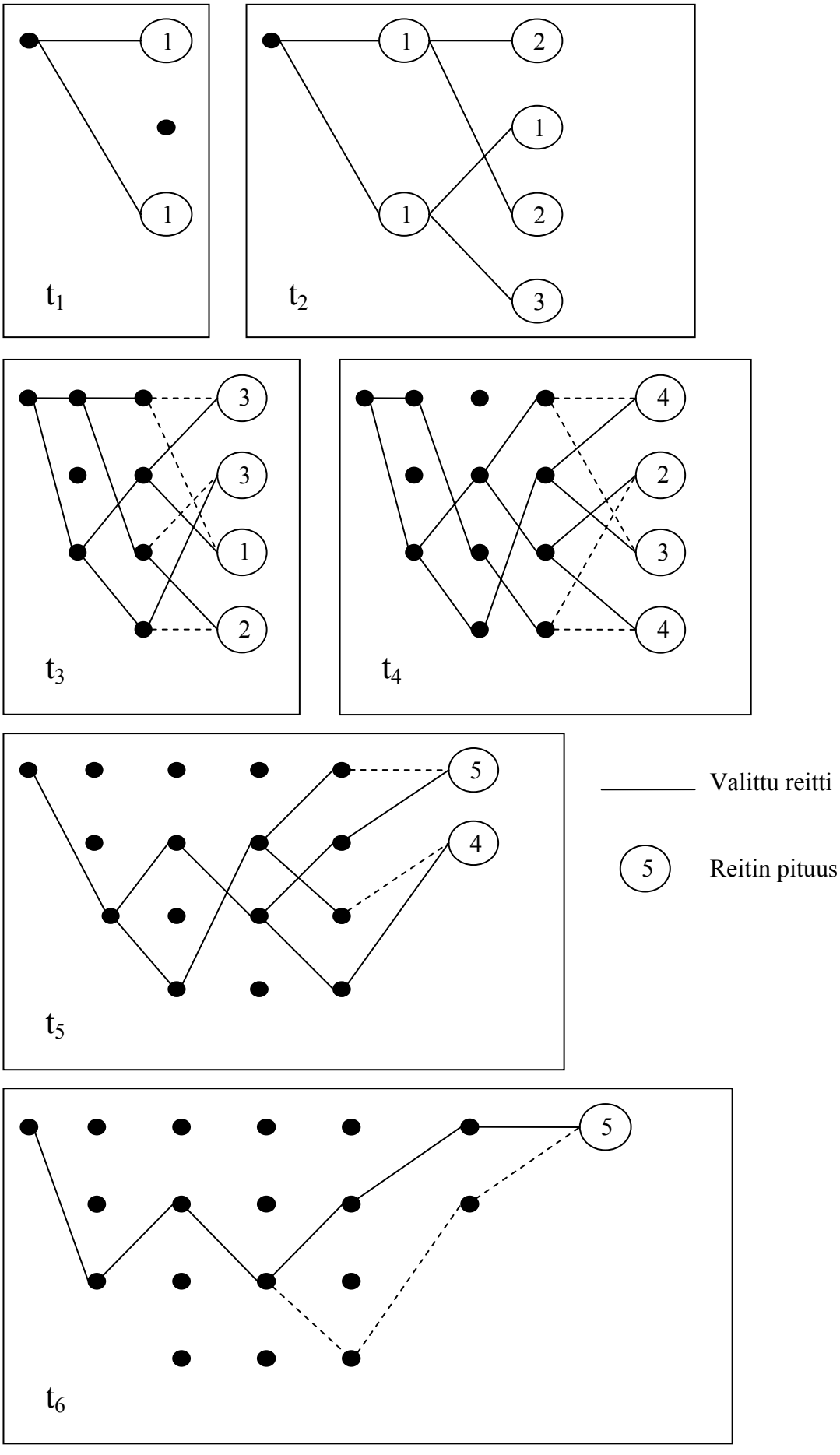


Kuvio 9: Ajanhetki t5.

Tilassa t5 uusien solmujen lukumäärä vähenee, samalla vähenee mahdollisten reittien lukumäärä. Loppua kohti tultaessa tulee tehdä enemmän valintoja reittien välillä, usein kuitenkin selviytyjäreitti on helppo valita. Tässä tapauksessa tultaessa tilaan tai ajanhetkeen t6, on mahdollisia reittejä jäljellä enää kaksi. Kahden yhtä pitkän reitin välillä algoritmi tekee valinnan, ja toinen reiteistä poistetaan. Jäljelle jäävä reitti on Viterbi-reitti. Jos on mahdollista, toinen yhtä pitkä reitti säilytetään muistissa.



Kuva 10: Ajanhetki t6, joka päättää esimerkissä tapahtuneen koodinpurun.


1
1
t1
1
2
1
1
2
3
t2
3
3
1
2
t3
4
2
3
4
t4
5
4
t5
Valittu reitti
5
Reitin pituus
5
t6

4.5 Käytännön esimerkki Viterbi-algoritmin todennäköisyyslaskennasta

Edellinen esimerkki Viterbi-algoritmin toimintaperiaatteesta ei vielä selventänyt, mistä todennäköisyydet eri tilojen välille lasketaan. Tästä voidaan ottaa seuraavanlainen esimerkki:

Kuvitellaan pariskunta Paavo ja Lisa. Paavo asuu Suomessa kesän vaihtelevassa ilmastossa ja Lisa asuu kaukana Australiassa jatkuvassa auringonpaahteessa. Lisa on lähiaikoina muuttamassa Suomeen ja häntä kiinnostaa pohjoinen ilmasto. Paavo ei uskalla kertoa sääoloja suoraan Lisalle, vaan hän kertoo puhelimessa, mitä on minäkin päivänä tehnyt. Paavon aktiviteetteihin kuuluvat jalkapallo, golf, kuntosali ja siivous. Nämä ovat Lisalle tarkasteltavia tiloja, joiden perusteella hänen tulee saada selville piilotetut tilat eli sääolot.

Lisa katsoo Suomen sääolojen esiintymistodennäköisyydet kesäajalle ja sään muutostodennäköisyydet seuraavalle päivälle. Hän saa näistä tiedoista seuraavanlaiset taulukot.

Taulukko 3: Eri sääolojen esiintymistodennäköisyydet

Tilojen todennäköisyydet	
Aurinkoinen	0,47
Pilvinen	0,20
Sateinen	0,33

Taulukko 4: Säätilojen muutostodennäköisyydet päivän vaihtuessa

δ_V		Sää tänään		
		Aurinkoinen	Pilvinen	Sateinen
Sää eilen	Aurinkoinen	0,400	0,350	0,250
	Pilvinen	0,325	0,200	0,475
	Sateinen	0,275	0,450	0,275

Sääolojen todennäköisyydet eivät kuitenkaan vielä riitä Lisalle, jotta hän tietäisi, minä päivänä on ollut millainen sää. Hänen täytyy yhdistää todennäköisyyksiin myös yhteys Paavon kertomiin aktiviteetteihin. Paavon aktiviteetteja eli tarkasteltavia tiloja on neljä. Mahdollisia piilotettuja tiloja on kolme. Näiden välisistä todennäköisyyksistä Lisa kasaa taulukon (Taulukko 5) sen tiedon perusteella, mitä hän tietää tai arvelee Paavon eri sääoloissa harrastavan.

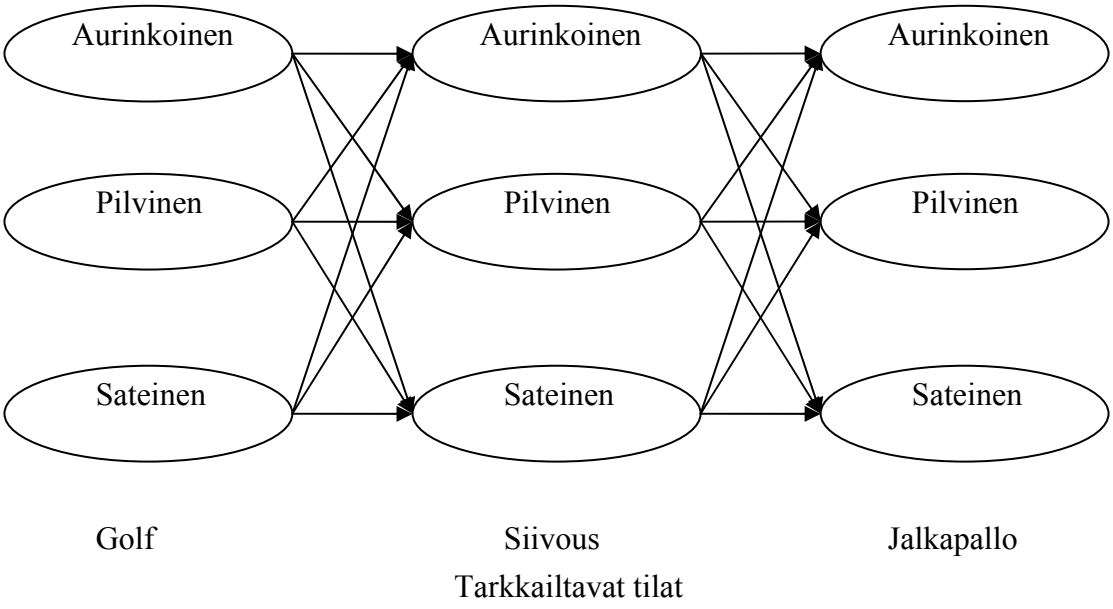
Taulukko 5: Vallitsevan sääolon todennäköisyys Paavon eri aktiviteeteissa.

δ_t	Jalkapallo	Golf	Kuntosali	Siivous
Aurinkoinen	0,30	0,60	0,09	0,01
Pilvinen	0,30	0,10	0,35	0,25
Sateinen	0,30	0,05	0,30	0,35

Taulukoissa esiintyvät desimaaliarvot kuvaavat todennäköisyyksiä prosenttiosuuksina 1 ollessa 100 % ja 0 ollessa 0 %.

Nyt Lisa on saanut selville tarvittavat todennäköisyydet, jotka aiemmassa esimerkissä oli annettu etäisyyksinä eri solmujen välillä. Lisalta puuttuu kuitenkin vielä tieto siitä, mitä Paavo on tehnyt eri päivinä. Tämän tiedon, eli tarkasteltavat tilat, Lisa saa selville, kun Paavo kertoo niistä puhelimessa. Paavo kertoo kolmen päivän aktiviteettiensa olleen järjestyksessä golf, siivous ja jalkapallo.

Kolmen päivän tarkkailtavien tilojen perusteella tämän esimerkin Trellis-kaavio on mahdollista asettaa kuvion 11 mukaiseen muotoon. Trellis-kaaviosta nähdään eri piilotettujen tilojen väliset yhteydet ja laskennan edetessä myös tarkkailtavien tilojen vaikutus Viterbi-reittiin.



Kuvio 11: Esimerkin Trellis-kaavio

Nyt kun Lisa on saanut tietoonsa kaiken paitsi varsinaisen halutun sekvenssin, hän antaa Viterbi-algoritmin hoitaa lopun. Algoritmi saa selville todennäköisimmät vallinneet säätilat niille kolmelle päivälle, joista Paavo on kertonut. Usein Viterbi-algoritmi tietää, mistä tilasta koodauksen purku pitää aloittaa, mutta tässä tapauksessa se on epäselvää. Tästä syystä algoritmi laskee ensimmäiseksi aloitustodennäköisyydet jokaiselle piilotetulle tilalle. /4/

Aloitustodennäköisyydet eri tiloille voidaan laskea seuraavanlaisella kaavalla:

$$\delta_x = \delta_e \cdot \delta_t \qquad (1),$$

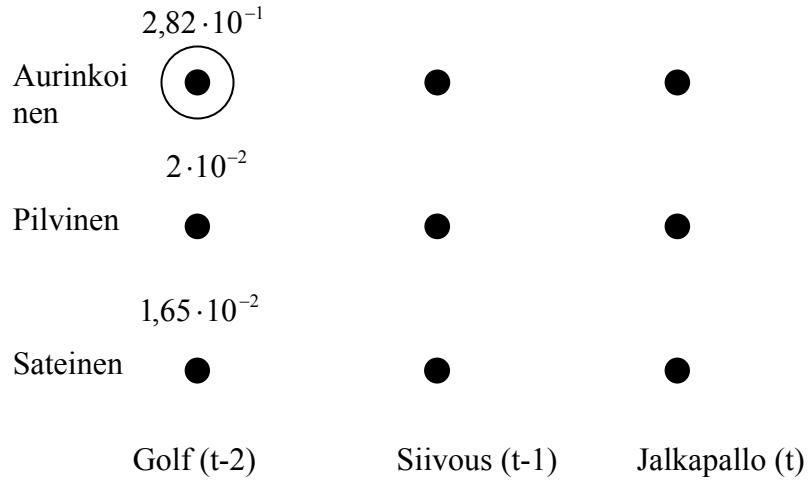
jossa

δ_x = todennäköisyys tarkkailtavassa tilassa kysytyllä ajanhetkellä,

δ_e = kyseisen piilotetun tilan esiintymistodennäköisyys,

δ_t = kyseisen piilotetun tilan todennäköisyys tarkkailtavassa tilassa.

Aloitustodennäköisyydet on laskettu kuviossa 12 näkyvään Trellis-kaavioon.



Kuvio 12: Trellis-kaavio, johon on laskettu aloitustodennäköisyydet kaikille piilotetuille tiloille ajanhetkellä t-2

Aloitustodennäköisyydet laskettuaan Viterbi-algoritmi saa selville todennäköisimmän aloitustilan. Tämä aloitustila on aurinkoinen. Seuraavaan on laskettuna esimerkin vuoksi sateinen-tilan aloitustodennäköisyys kaavalla 1.

$$\delta_x = \delta_e \cdot \delta_t$$
$$= 0{,}33 \cdot 0{,}05 = \underline{0{,}0165}$$

Lukuarvot kyseistä laskua varten on merkitty taulukoihin 3 ja 4.

Seuraavaksi aloitustodennäköisyyksien laskennan jälkeen on vuorossa seuraava päivä (t-1), jolloin Paavo kertoi siivonneensa. Toisin kuin aloitustilaa laskiessa, ajanhetkellä t-1 tulee laskennassa ottaa huomioon aiempi tila. Trellisissä edetessä tulee siis jatkossa käyttää seuraavanlaista kaavaa

$$\delta_x = \max((\delta_{a1} \cdot \delta_{v1}),(\delta_{a2} \cdot \delta_{v2}),(\delta_{a3} \cdot \delta_{v3})) \cdot \delta_t \qquad (2),$$

jossa

δ_{a1} = Aiemman tilan todennäköisyys. a1 = aurinkoinen, a2 = pilvinen ja a3 = sateinen

δ_{v1} = Vaihtumistodennäköisyys aiemmasta tilasta laskennassa kysyttyyn tilaan. v1 = aurinkoisesta tilasta, v2 = pilvisestä tilasta ja v3 on sateisesta tilasta.

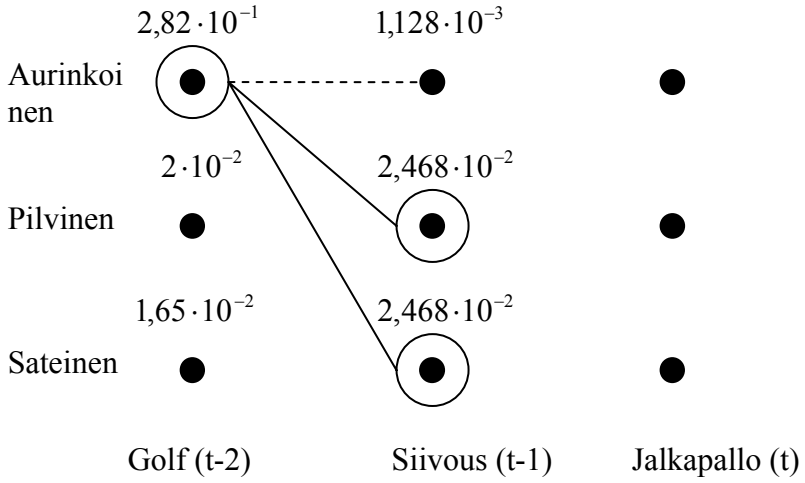
Esimerkkinä kaavan 2 käyttämisestä saadaan seuraavanlaista:

1) Käytetään aiemmin laskettuja todennäköisyyksiä piilotetuille tiloille ajanhetkellä t-2. Nämä arvot nähdään kuviosta 12.
2) Katsotaan taulukosta 3 säätilan vaihtumistodennäköisyys.
3) Katsotaan jokaiselle piilotetulle tilalle sen todennäköisyys tarkasteltavassa tilassa taulukosta 4 ja sijoitetaan nämä arvot kaavaan. Alla on laskettu pilvinen-tilan todennäköisyys ajanhetkellä t-1.

$$\begin{aligned}\delta_x &= \max((\delta_{a1}\cdot\delta_{v1}),(\delta_{a2}\cdot\delta_{v2}),(\delta_{a3}\cdot\delta_{v3}))\cdot\delta_t \\ &= \max((0{,}282\cdot 0{,}35),(0{,}02\cdot 0{,}2),(0{,}0165\cdot 0{,}45))\cdot 0{,}25 \\ &= 0{,}0987\cdot 0{,}25 \approx \underline{2{,}468\cdot 10^{-2}}\end{aligned}$$

Kaavaa käyttäessä tulee ottaa huomioon, mistä aiemmasta tilasta mikäkin laskettu todennäköisyys tulee. Tässä tapauksessa tiedetään, että todennäköisimpiä aloitustiloja on vain yksi eli aurinkoinen.

Laskemalla Trellis-kaaviota eteenpäin kaavalla 2, saadaan todennäköisyydet eri piilotetuille tiloille ajanhetkellä t-1 kuvioon 13.

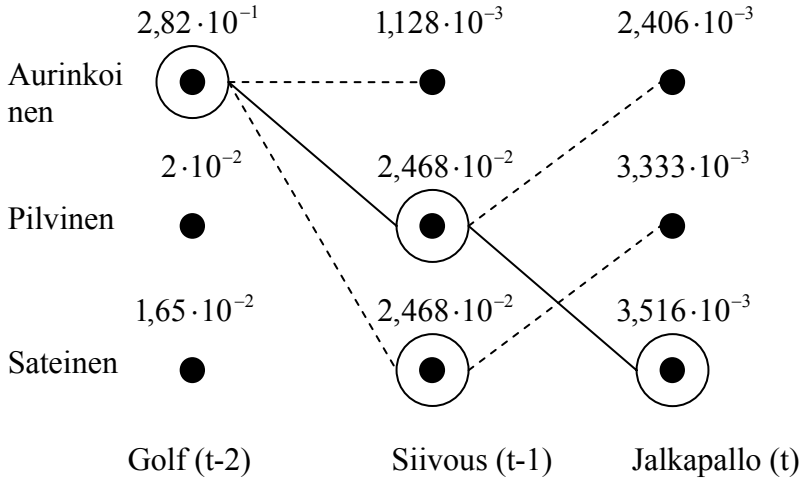


Kuvio 13: Viterbi-reitin eteneminen Trellisissä ajanhetkeen t-1.

Kuten kuviosta 13 nähdään, kahdelle piilotetulle tilalle sattuu tulemaan sama todennäköisyys ajanhetkellä t-1. Yksi tiloista on selvästi muita epätodennäköisempi, ja siksi siihen kulkeva reitti poistetaan reittihistoriasta ja keskitytään vain kahteen todennäköisempään reittiin. Mikäli kahdelle eri reitille olisi tullut sama todennäköisyys ja ne olisivat johtaneet samaan solmuun, olisi toinen niistä voitu poistaa. Tässä tapauksessa kuitenkin reitit johtavat eri solmuihin, eikä voida vielä tietää, kumpi reiteistä on Viterbi-reitti.

Kaavaa 2 voidaan käyttää koko lopun Trellis-kaavion läpi laskemiseen. Tarkkailtavia tiloja voisi olla suuriakin määriä, eikä laskenta muuttuisi tämän monimutkaisemmaksi. Reittejä tulisi mahdollisesti säilöä muutamia laskennan edetessä pidemmälle, mutta vain kolmen piilotetun tilan Trellis-kaaviossa ei kovin monia eri reittivaihtoehtoja ole. Suuremmissa kaavioissa voi kuitenkin muistikapasiteetin tarve tulla ongelmaksi.

Kuvioon 14 on havainnollistettu esimerkin valmis Viterbi-reitti todennäköisyyksineen. Reitti on esitetty yhtenäisellä viivalla, ja algoritmin käymät muut reitit on vielä pidetty tallessa katkoviivoilla.

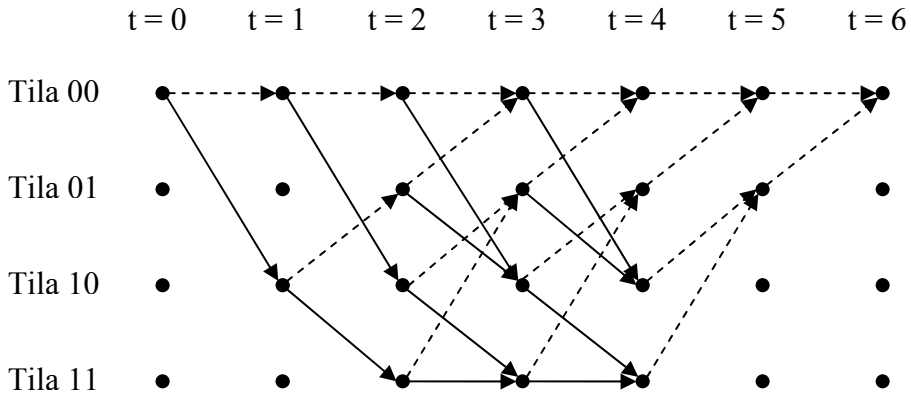


Kuvio 14: Valmis Viterbi-reitti

Täten valmiiksi Viterbi-reitiksi saadaan sekvenssi järjestyksessä aurinkoinen, pilvinen, sateinen. Viterbi-algoritmin ja tarkkailtavien tilojen avulla on todennäköisesti saatu selville vallinneet sääolot kolmen päivän ajalta. Reitin laskentaa voitaisiin jatkaa paljon pidemmällekin, mutta silloin tarvittaisiin myös uusien päivien tarkkailtavat tilat. Myös tässä esimerkissä olisi voinut olla tarpeen säilyttää suurempaa reittihistoriaa, koska ajanhetkellä t-1 pilvinen ja sateinen saavat saman todennäköisyyden seuraavaksi tilaksi.

4.6 Viterbi-algoritmi koodinpurkajana konvoluutiokoodatussa lähetteessä

Seuraavaksi käydään läpi esimerkki Viterbin toiminnasta konvoluutiokoodatun lähetteen purkamisessa. Tämän esimerkin havainnollistamisessa käytetään Trellis-kaaviota, kuten aiemmissakin esimerkeissä on käytetty. Otetaan neljätilainen Trellis-kaavio, jossa tilat ovat 00, 01, 10 ja 11. Konvoluutiokoodaaja lähettää neljän bitin viestin, jossa on virhe. Kuviosta 15 nähdään esimerkin Trellis-kaavio. /5/



Kuvio 15: Esimerkin Trellis-kaavio

Neljän bitin viestin lisäksi kaaviossa näkyvät kaksi koodaajan muistibittiä, joten tiloja tulee kuusi aloitustilan t = 0 lisäksi. Yhtenäiset viivat kuvaavat tilojen muutoksia, kun koodaajan sisäänmenobitti on yksi. Katkoviivalla on merkitty ne muutokset, jotka tapahtuvat sisäänmenobitin ollessa nolla. Tilat voivat muuttua tietyn tavan mukaisesti. Tilojen muutokset ja koodaajan antamat kanavasymbolibitit nähdään taulukoista 6 ja 7.

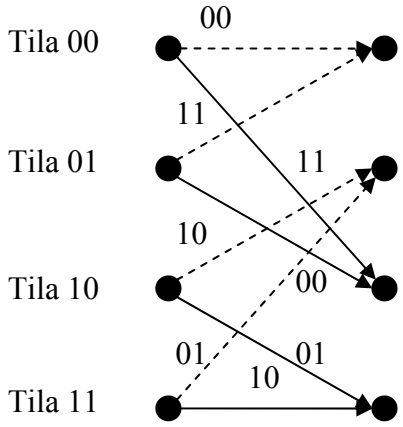
Taulukko 6: Tilojen muutokset

	Seuraava tila, jos	
Nykyinen tila	Sisäänmeno = 0	Sisäänmeno = 1
00	00	10
01	00	10
10	01	11
11	01	11

Taulukko 7: Kanavasymbolit

	Ulostulosymbolit	
Nykyinen tila	Sisäänmeno = 0	Sisäänmeno = 1
00	00	11
01	11	00
10	10	01
11	01	10

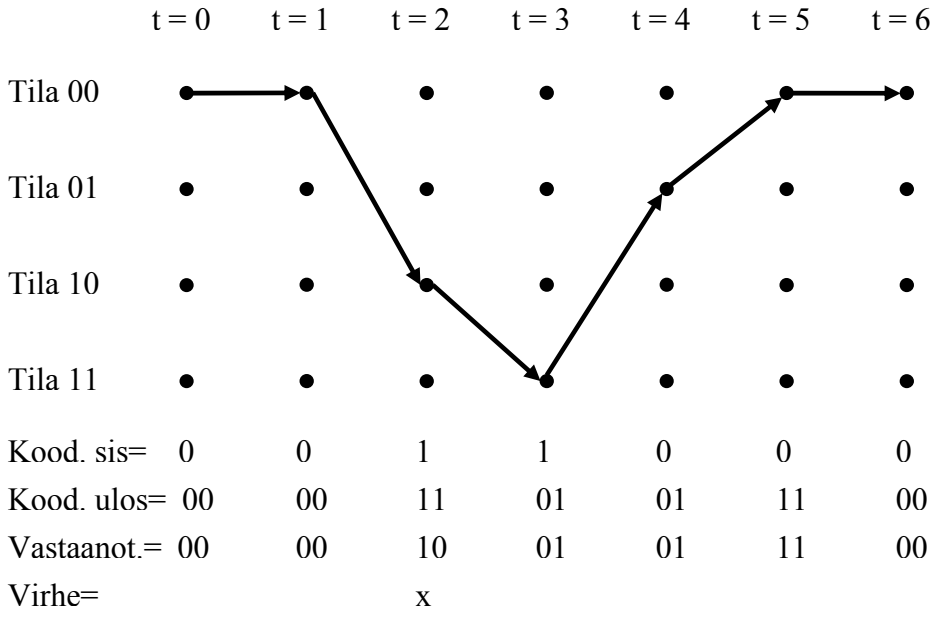
Tilamuutokset voidaan havainnollistaa myös tarkemman kuvan avulla. Kuviossa 16 on osa Trellis-kaaviota ja kuvasta nähdään tilamuutokset ja tilamuutoksien aiheuttamat kanavasymbolibitit. Tässäkin kuvassa katkoviivat tarkoittavat tilamuutoksia, kun sisäänmenobitti on nolla, ja yhtenäinen viiva tilamuutoksia, joissa sisäänmenobitti on yksi. Kanavasymbolibitit on sijoitettu viivojen viereen. /5/



Kuvio 16: Tilamuutokset ja kanavasymbolibitit

Seuraavaksi voidaankin tutkia, miten Viterbi-algoritmi toimii. Tätä varten tarvitaan lähete eli koodaajan sisäänmenoon menevät neljä bittiä, joita Viterbi-algoritmi yrittää koodista purkaa. Nämä neljä bittiä ovat tässä tapauksessa 0, 1, 1 ja 0. Koodaaja on asetettu lähetyksen alussa nollatilaan, joten aloitustila ajanhetkellä t = 0 on 00. Tämän jälkeen tulee lähete ja lähetteen perässä on muutama muistintyhjennysbitti. Nämä muistintyhjennysbitit ovat nollia, ja niiden avulla koodaaja saadaan jälleen tilaan 00.

Viestinä lähetetään siis 0, 1, 1 ja 0. Tämä nähdään kuviossa 17. Kuvioon on myös lisätty koodaajan ulosantamat kanavasymbolibitit ja vastaanottopuolen vastaanottamat symbolibitit. Samalla kuviossa näkyy Viterbi-reitti lähetteelle.



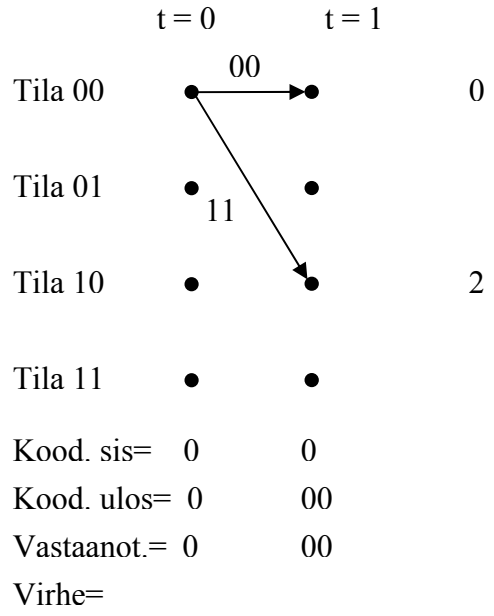
Kuvio 17: Valmis Viterbi-reitti annetulle lähetteelle

Kuvioon 17 on merkitty virhe, jonka Viterbi-algoritmi havaitsee koodinpurun aikana. Koodaajasta ulostulevat symbolibitit voidaan muodostaa taulukon 7 avulla.

Joka kerta, kun vastaanotetaan kanavasymbolipari, Viterbi-algoritmi laskee etäisyyden vastaanotetun bittiparin ja kaikkien mahdollisten bittiparien välille. Aloitustilana koodaajalla on 00 ajanhetkellä t = 0. Tämän takia ainoat kanavasymboliparit, jotka voidaan vastaanottaa mentäessä ajanhetkestä t = 0 ajanhetkeen t = 1, ovat 00b ja 11b. Jos sisäänmenoksi koodaajaan menee 0, tulee ulostulona 00b, kun sisään menee 1, tulee ulostulona 11b. /5/

Viterbi-algoritmi käyttää laskennassa ns. Hamming-etäisyyttä (eng. Hamming distance), joka lasketaan yksinkertaisesti vertaamalla, kuinka monta bittiä on eroa vastaanotettujen bittiparien ja kaikkien mahdollisten bittiparien välillä. Tulos voi olla nolla, yksi tai kaksi. Tästä muodostuu samalla tavalla reitin pituus, kuten aiemmissa esimerkeissä on nähty. Hamming-etäisyys lasketaan jokaisella ajanhetkellä, ja edelliset reittien pituudet tai siihenastiset etäisyydet pidetään muistissa.

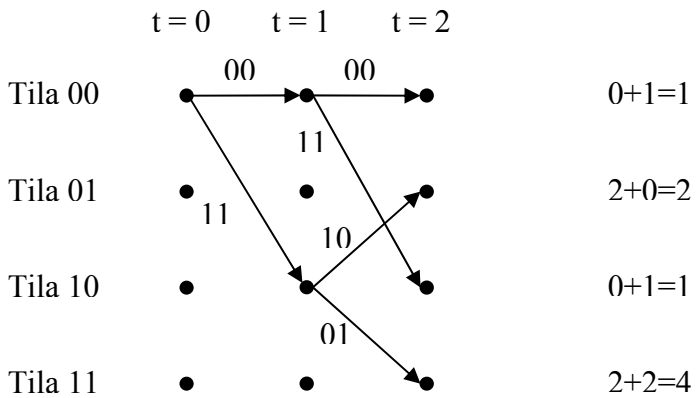
Ajanhetkellä t = 1 purkaja vastaanottaa 00b. Ainoat mahdolliset kanavasymboliparit, joita olisi voitu vastaanottaa, olivat 00b ja 11b. Hamming-etäisyys 00b ja 00b välillä on nolla. Hamming-etäisyys 00b ja 11b välillä on kaksi. Tämän takia oksa tai reitti tilasta 00 tilaan 00 on nolla ja reitti tilasta 00 tilaan 10 on kaksi. Aiempia reittipituuksia ei vielä ole tallennettu, joten nämä luvut ovat tämänhetkiset reittien pituudet. Kuvio 18 havainnollistaa tilanteen.



Kuvio 18: Ajanhetki t = 1

Muodostuvat reitit tallennetaan muistiin, koska ei vielä tiedetä, kumpi reiteistä on oikea. Reittejä ei tallenneta graafisesti kuten kuvassa, vaan numeraalisesti. Reiteistä tallennetaan aiempi tila ja reitin pituus tultaessa uuteen tilaan.

Siirryttäessä tilaan t = 2, vastaanotetaan bittipari 10b. Ajanhetkellä t = 2 on mahdollista vastaanottaa mikä tahansa kanavasymbolipari, koska reittien määrä lisääntyy. Tilasta 00 mentäessä tilaan 00 koodaaja lähettää parin 00b. Mentäessä tilasta 00 tilaan 10 lähetetään 11b. Tilasta 10 mentäessä tilaan 01 lähetetään 10b ja 01b lähetetään mentäessä tilasta 10 tilaan 11. Kuvioon 19 on laskettu kaikkien reittien pituudet yhdistämällä uuden oksan pituus aiempaan tallennettuun pituuteen.

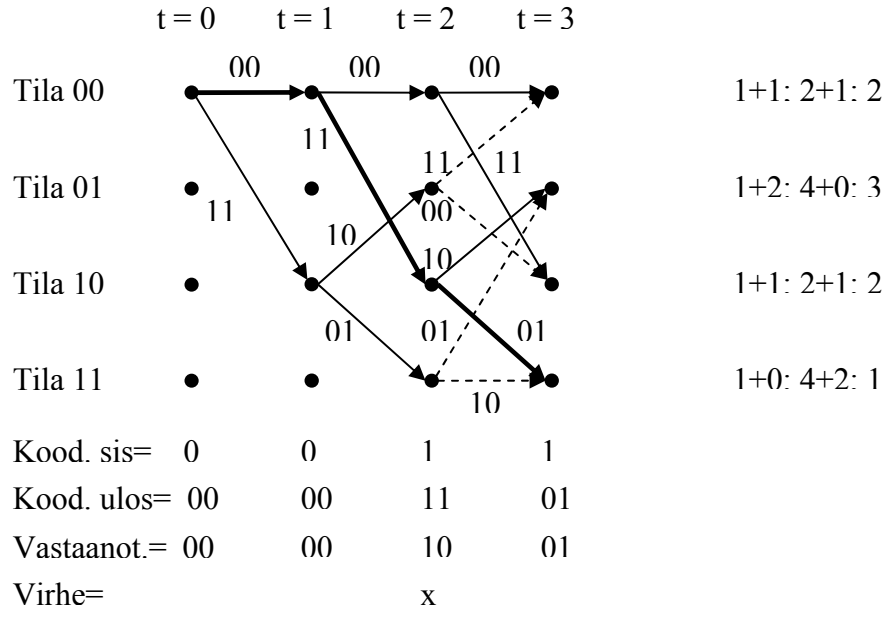


Kood. sis=	0	0	1
Kood. ulos=	0	00	11
Vastaanot.=	0	00	10
Virhe=			x

Kuvio 19: Ajanhetki t = 2

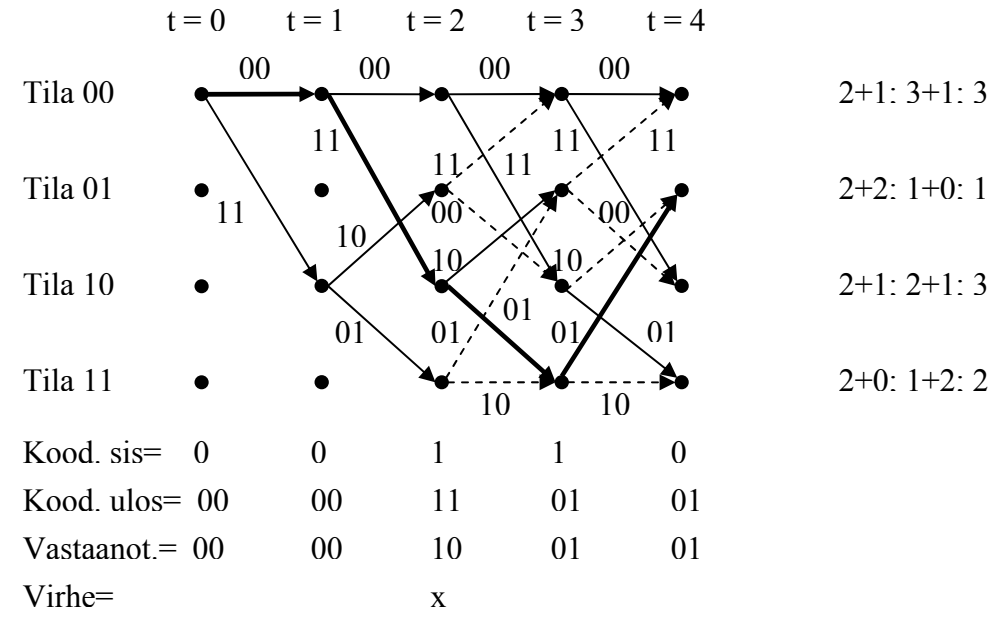
Reittipituuksia laskettaessa havaitaan, että Viterbi-algoritmi saa lyhimmän reitin arvoksi 1. Tästä voidaan suoraa päätellä yhden bitin virheen tapahtuneen jossain vaiheessa lähetettä. Virhe nähdään, kun verrataan koodaajasta uloslähtevää bittiparia ja vastaanotettua bittiparia ajanhetkellä t = 2. Viterbi-algoritmi ei kuitenkaan tiedä, mitä koodaaja on lähettänyt, sillä vain vastaanotetut bitit ovat tiedossa. Vaikka osa reiteistä on selvästi muita pidempiä jo tässä vaiheessa, säilytetään jokaista tilaa kohden yksi siihen johtava reitti.

Kuten kuviosta 20 nähdään, reittivalinta muuttuu hiukan monimutkaisemmaksi tultaessa ajanhetkeen t = 3. Uusia reittejä tulee kaksi jokaisesta neljästä edellisestä tilasta neljään uuteen tilaan. Uusiin tiloihin tulee täten samanaikaisesti kaksi reittiä jokaiseen. Laskenta ei sinällään muutu yhtään vaikeammaksi, sillä Viterbi-algoritmi vertaa jälleen jokaisen uuden oksan kanavasymboliparia vastaanotettuun pariin ja määrää sille Hamming-etäisyyden avulla pituuden. Kuviossa oikealle on laskettu reittien pituudet. Aiempaan reittipituuteen on summattu uuden oksan pituus. Ylempi oksista on laskettu ensin. Näitä kahta tulosta vertaamalla valitaan pienempi reitti ja toinen katkaistaan. Katkaistu reitti näkyy katkoviivalla. Samanpituisten reittien tullessa samaan tilaan käytetään tässä tapauksessa satunnaista valintaa reittien välillä.

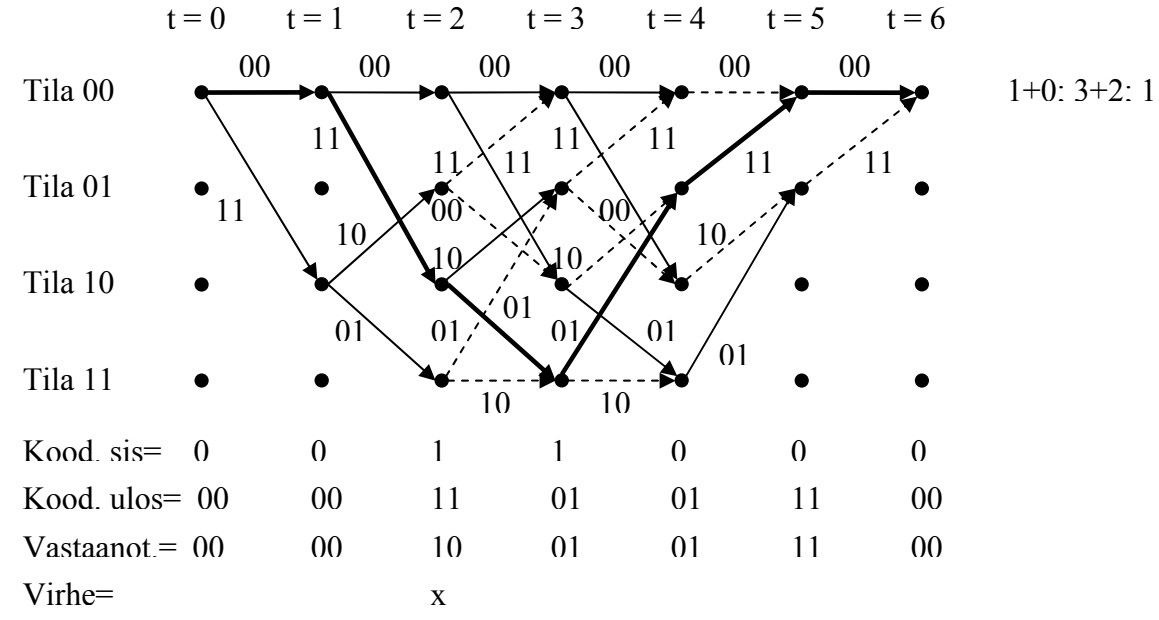


Kuvio 20: Ajanhetki t = 3

Laskennan jälkeen jäljelle jäävät reittipituudet jokaisessa tilassa. Paksumpi viiva kuvissa tarkoittaa Viterbi-reittiä. Sen pituus on 1, koska reitissä oli yksi virhe. Ajanhetkeen t = 4 (kuvio 21) mentäessä laskenta jatkuu samalla tavalla ja lähetetty viesti saapuu perille lukuun ottamatta muistintyhjennysbittejä, jotka saadaan mukaan kuviossa 22.



Kuvio 21: Ajanhetki t = 4



Kuvio 22: Trellis-kaavio annetulle lähetteelle

Viestin perään liitetyt kaksi muistintyhjennysbittiä saattavat koodaajan jälleen tilaan 00. Viterbi-reitti saadaan selville ja sen pituudeksi tulee 1. Toisen loppuun asti pääsevän reitin pituudeksi tulee 5, joten oikean reitin valinta on helppoa. Tämän Viterbi-reitin avulla purkaja pystyy palauttamaan tietoon sen lähetteen, joka alun perin koodaajan sisäänmenoon annettiin.

Viterbi-algoritmi on näin selvittänyt Viterbi-reitin kyseiselle lähetteelle, mikä tarkoittaa, että alkuperäinen koodi saadaan näiden tietojen avulla purettua. Viterbi-algoritmi on nykyisille laitteistoille varsin yksinkertainen algoritmi suoritettavaksi. Suurimman ongelman Viterbi-algoritmissa tuottaa reittihistorian ylläpito. Viterbi-algoritmi ei sellaisenaan ole omiaan pitkille viesteille, koska pitkien viestien purussa tulee myös purkuikkunan kokoa kasvattaa. Purkuikkunan koon kasvattaminen puolestaan lisää muistissa olevien reittien määrää ja siten muistikapasiteetin tarve lisääntyy.

5 Muita algoritmeja

Monet nykyiset algoritmit käyttävät eteenpäin-taaksepäin-algoritmin periaatetta, kun lasketaan tiettyjen tilojen esiintymistodennäköisyyksiä tarkkailtavan sekvenssin avulla. Eteenpäin-taaksepäin algoritmi laskee todennäköisyydet tiloista eteenpäin ja taaksepäin. Tämänkaltaisiin algoritmeihin kuuluvat mm. jo käsitelty Viterbi-algoritmi ja Baum-Welch-algoritmi. Koodinpurussa on käytössä monia algoritmeja, joista käydään seuraavaksi muutama läpi pintapuolisesti.

5.1 Fano-algoritmi

Fano-algoritmi toimii lähes samalla periaatteella kuin Viterbi-algoritmi. Fano-algoritmi etsii todennäköisintä reittiä Trellis-kaavion läpi saadakseen selville alun perin lähetetyn viestin. Fano-algoritmi poikkeaa Viterbi-algoritmista kuitenkin siinä, että se käy Trellis-kaaviota läpi reitti kerrallaan. Tällä tavalla Fano-algoritmin virheenkorjauskyky on samaa luokkaa kuin Viterbi-algoritmilla. Etuina Fano-algoritmilla on pienempi muistikapasiteetin tarve ja mahdollisuus pidempiin viestikehyksiin. Haittapuolena viive on koodinpurussa paljon suurempi. /2, s. 25/

5.2 Baum-Welch-algoritmi

Baum-Welch-algoritmi on myös samankaltainen kuin Viterbi-algoritmi. Sitä käytetään löytämään tuntemattomat parametrit Markovin piilomallissa. Se hyödyntää eteenpäin-taaksepäin-algoritmia ja on nimetty Leonard E. Baumin ja Lloyd R. Welchin mukaan. Baum-Welch-algoritmi laskee jokaiselle tilalle todennäköisyyden eteenpäin ja taaksepäin. Sen laskenta tilamuutoksien välillä perustuu myös todennäköisyyksiin. /6/

6 Sovellukset ja käyttökohteet

Konvoluutiokoodauksen ja erilaisten koodinpurkaja-algoritmien sovelluksia ja käyttökohteita on nykyisin lähes lukematon määrä. Kun Viterbi-algoritmi tuli esille vuonna 1967, oli se Andrew J. Viterbin asianajajan mielestä aivan liian raskas käytettäväksi kenenkään muun kuin Yhdysvaltojen valtion käytössä. Tästä syystä algoritmi jäi aikoinaan patentoimatta. Nykyisin kuitenkin lähes jokainen ihminen on käsitellyt jotain laitetta, jossa Viterbi-algoritmi on purkamassa tulevaa digitaalista tietoa.

Viterbi-algoritmi on käytössä tällä hetkellä yli kahdessa miljardissa puhelimessa. Viterbi-algoritmin sanotaan mullistaneen koko televiestinnän alan. Sitä käytetään digitaalisissa televisiojärjestelmissä purkamaan tulevaa tietoa. Monia muita vasta kehitteillä olevia sovelluksia on paljon. Purkualgoritmeja on suunniteltu puheentunnistus- ja tekstintunnistuskäyttöön ja myös biotekniikan alalle sovelluksia on suunnitteilla. Markovin piilomallin ja Viterbi-algoritmin muunnelman avulla voidaan biotekniikan alalla mm. tunnistaa geenejä.

Puheentunnistus on tällä hetkellä yksi kehitellyimmistä ja kehittyvimmistä koodinpurun sovelluksista. Puheentunnistuksen periaate on saada tietotekniikka ja muut tekniset sovellukset toimimaan ihmisen puhekomentojen mukaan. Yksinkertaisimmillaan järjestelmä koostuu valmiiksi sovituista puhuttavista komennoista, joiden avulla voidaan tehdä rajattu määrä toimintoja. Ensimmäisiä laajimmalle levinneitä ohjelmia kyseiseen tarkoitukseen on mm. Microsoftin Speech Recognition, joka tulee Windows Vista-ohjelmiston mukana. Kyseisellä ohjelmalla suoritettavia toimintoja on jo runsaasti, mutta kielivalikoima rajoittuu englantiin. Suurin ongelma saattaakin olla erilaisien kielien huomioon ottaminen.

Monimutkaisemmissa sovelluksissa puheentunnistuksen päämäärä on kasata teksti ihmisen vapaasta puheesta. Tämä on hyvin haastavaa, koska ulostuleva ääni on jokaisella hyvin yksilöllinen. Ihmisen ääni on ilmanpaineen nopeaa värähtelyä.

Värähtely riippuu ihmisen yksilöllisistä ominaisuuksista kuten hengityksestä, kurkunpäästä, kielestä ja huulista. Vaikka nämä kaikki ominaisuudet otetaan purkavassa algoritmissa huomioon, tulee ongelmaksi myös se, että saman puhujan äänisignaali voi eri kerroilla olla erilainen.

Puheentunnistuksessa käytetään Markovin piilomallia. Markovin piilomalli saa sanojen todennäköisyydet muodostetusta kielimallista. Kielimalli on todennäköisyysfunktio odotettavissa olevalle sanalle aiempien sanojen perusteella. Kielimalli voidaan myös muodostaa sanan eri osille. Viterbi-algoritmille annetaan syötteenä foneemit eli kahden äänteen yhdistelmät, jotka koostuvat ensimmäisen äänteen jälkipuoliskosta ja seuraavan äänteen alkupuoliskosta.

7 Yhteenveto

Tässä työssä käsiteltiin konvoluutiokoodausta ja siihen liittyviä virheentunnistus- ja virheenkorjausmenetelmiä. Tämän työn alussa käytiin läpi Markovin malli, Markovin piilomalli ja niihin liittyvät rakenteet ja käsitteet, jotta konvoluutiokoodauksessa käytettyjen virheenkorjausmenetelmien toimintaa kyettäisiin ymmärtämään. Konvoluutiokoodauksen perusperiaate on, että yhtä bittiä kohden lähetetään useampi bitti kanavalle. Näitä bittejä kutsutaan kanavasymbolibiteiksi. Kanavasymbolibittien avulla yhden bitin kääntymisestä aiheutuva virhe saadaan suhteessa pienemmäksi.

Kaikkia virheitä konvoluutiokoodauksella ei pystytä poistamaan. Konvoluutiokoodaus auttaa kuitenkin poistamaan suurimman osan virheistä, ja tämä riittää siihen, että lähetys saadaan onnistuneesti perille. Purkupuolella tulee olla konvoluutiokoodauksen purkuun erikoistunut algoritmi. Pääasiallisena purkualgoritmina käytiin läpi Viterbi-algoritmi. Viterbi-algoritmi on varsin laajalti käytössä konvoluutiokoodattujen lähetteiden purussa. Algoritmi on käytössä yli kahdessa miljardissa matkapuhelimessa ympäri maailman ja sen kehittäjä, amerikanitalialainen insinööri Andrew J. Viterbi, oli ehdolla Millenium-palkinnon saajaksi vuonna 2008.

Viterbi-algoritmi purkaa konvoluutiokoodatun lähetteen todennäköisimmän reitin periaatteella. Purkaja käyttää samanlaista Trellis-kaaviota kuin koodauspuolella on käytetty. Koodaajan aloitustila on purkajalla tiedossa, ja koodaajan käyttäytymistä tarkkaillaan vastaanotettujen kanavasymbolibittien avulla. Kanavasymbolibiteistä saadaan todennäköisin reitti vertaamalla niitä kaikkiin mahdollisiin reitteihin ja selvittämällä tästä todennäköisin polku eteenpäin Trellis-kaaviossa. Reittejä voidaan pitää muistissa muistikapasiteetista riippuen hyvinkin paljon. Mitä enemmän reittejä pidetään muistissa, sitä varmemmin tulee oikea reitti valituksi ja lähete saadaan purettua alkuperäisessä muodossa.

Lähdeluettelo

Painetut lähteet

1 Andrew J. Viterbi 1995. CDMA, Principles of Spread Spectrum Communication. Addison Wesley Longman, Inc.

2 Kivioja Mikko 1997. DAB Specific Viterbi Decoder Implementation and Optimization Using VHDL.

3 Savolainen Marko 2001. Reusable Viterbi Decoder Implementation.

Sähköiset lähteet

4 Professori Roger D. Boylen tutoriaali liittyen Markovin piilomalliin. [viitattu 17.2.2009]. Saatavilla osoitteesta: http://www.comp.leeds.ac.uk/roger/HiddenMarkovModels/html_dev/main.html

5 Viterbi-tutoriaali lähetyksestä [viitattu 21.4.2009]. Saatavilla osoitteesta: http://home.netcom.com/~chip.f/viterbi/tutorial.html

6 IEEE Information Theory Society Newsletter, Vol. 53, No.4, 2003 [viitattu 6.4.2009]. Saatavilla osoitteesta: http://www-rcf.usc.edu/~lototsky/MATH508/Baum-Welch.pdf

7 Millenium-palkinnon uutissivut [viitattu 3.4.2009]. Saatavilla osoitteesta: http://www.millenniumprize.fi/news/82/502/d,news/