A konvolúciós kódolás

A blokk kódok vitathatatlan előnye a viszonylag egyszerű algebrai kezelhetőség. Ugyanakkor felmerülhet bennünk a gondolat, hogy nem jár-e hátránnyal az, hogy az üzenetszimbólumok sorozatában a kódolással létrejövő belső kötöttségek megszakadnak a kódzsavak határain. Vagy más nézőpontból fogalmazva, arra is gondolhatunk, hogy a kódsebesség a kódszavak hosszával egyre nő, és azonos képességű kódok (például a bevezető példánkban szereplő Hamming kód) a kódszóhosszal növekvő kódolási nyereséget mutatnak. Nem lehetne végtelen hosszú kódszavakat használni?

A kérdésre a konvolúciós kód ad pozitív választ abban az értelemben, hogy elkezdhetünk kialakítani egy kódszót, amikor meg akarjuk kezdeni az információtovábbítást, és folyamatosan tarthat a kódszó, ameddig van továbbítandó információnk. Ezáltal persze nem lesz végtelen, de olyan modellekkel fogunk foglalkozni, amelyek egyik irányban végtelen hosszú sorozatokat tételeznek fel.

Hogyan alakítjuk ki tehát a konvolúciós kódot? Vesszük a továbbítandó szimbólumsorozat K hosszú szegmensét, képezünk hozzá egy $N > K$ hosszú sorozatot, mint a kódszó egy részét, majd vesszük a következő K adatszimbólumot, de ebből oly módon képezzük a kódszó következő N elemét, hogy tekintetbe vesszük a korábbi üzenetszimbólumokat is, vagy pontosabban azok véges és kötött hosszúságú szakaszát. Tehát miként egy lineáris invariáns transzformáció az időtartományban konvolúciót képez a bemenő jel és a súlyfüggvény között, létrehozva így a kimenő jelet, akként a konvolúciós kódoló is végigcsúsztatja egy speciális digitális szűrő súlyfüggvényét az adatszimbólumokon és előállítja a kódszimbólumokat. A specialitása abban rejlik, hogy több szimbólumot generál, mint amennyit a bemenetén feldolgoz.

A konvolúciós kódok egy nagyobb családnak, a *trellis* kódoknak azt a részét képezik, amelyek időinvariánsak, és lineárisak. A trellis egy rácsalakú irányított gráf, amelynek jelentős szerepe lesz a konvolúciós kód megértésében, ezért majd részletesebben megismerjük. Az időinvariancia röviden azt jelenti, hogy egy szimbólumsorozatra kapott kód nem változik meg attól, hogy eléje egy csupa nulla sorozatot helyezünk.

A konvolúciós kódok is, miként a blokk-kódok lehetnek szisztematikusak vagy nem szisztematikusak.

A konvolúciós kódolás leírása történhet trellis-ekkel, polinomokkal, vagy mátrixokkal. A viszonylagos egyszerűsége, a dekódolásban és a megvalósításban játszott szerepe miatt a trellis leírásra fogjuk a hangsúlyt helyezni.

Konvolúciós kódolás leírása mátrixokkal

Lényegében csak az érdekesség kedvéért mutatjuk meg, hogy a konvolúciós kódolás művelete mátrix-műveletekkel is leírható, bár a használandó mátrixok elég speciálisak. Nevezetesen végtelen sor- és oszlopszámú mátrixokkal való szorzás révén hozhatjuk létre a korlátlan hosszú kódszót a korlátlan hosszú adatsorozatból. Ebből egyenesen következik a gyakorlati használhatatlansága a mátrix leírásnak, de az elvégzendő műveletekre jól rávilágít.

Legyen példaként egy (3,1)-es konvolúciós kód, az alábbi generátormátrixszal:

$$\mathbf{G} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & \cdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & \cdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

A generátormátrix elemeit olyan rész-mátrixok adják, amelyek K sort és N oszlopot tartalmaznak, tehát a példában 1 soros és 3 oszlopos rész-mátrixokat látunk, amelyből az elsőt bekereteztük. A végtelen méretű generátormátrix elemei jobbára nullák, kivéve az átló környékén lévő elemeket, akkora szélességben, mint a kód *kényszer-hossza*. A kód kényszerhossza azt mondja meg, hogy hány *adatkereten* keresztül jön létre kapcsolat a kódszóban. Azt is mondhatjuk, hogy milyen „hosszú súlyfüggvénnyel" képezzük a konvolúciót. A mátrix közepe táján bekereteztük a részmátrixoknak azt a csoportját, amely végigvonul a teljes mátrix átlója mentén, és a következő pontban még kiemelt szerepe lesz.
A példát tekintve, legyen egy üzenetsorozatunk az alábbi:

$$\mathbf{u} = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & \cdots \end{bmatrix},$$

a keletkező kódszót a következő szorzat adja:

$$\mathrm{c} = \mathbf{u} * \mathbf{G} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & \cdots \end{bmatrix}.$$

Bár igen gyakran a K=1 hosszú adatkeretű konvolúciós kódokat használják, egyáltalán nem kizárt a hosszabb adatkeret alkalmazása, amire szintén mutatunk egy példát. Legyen a kód (3,2)-es, a következő generátormátrixszal:

$$\mathbf{G} = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & \cdots \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & \cdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & \cdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

A generátormátrixot eredményező rész-mátrixok sorai leírják az adatszimbólumokból képezendő kódszimbólumokat. Például egy csupa 1-es üzenet-sorozatból a következő kód-sorozat keletkezik:

$$\mathrm{c} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} * \mathbf{G} = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}.$$

A generátormátrixon kívül megszerkeszthető a paritásellenőrző mátrix, és az ú.n. inverz mátrix, amely visszaadja a kódsorozatból az üzenetsorozatot. Ezek szerepe azonban nem jelentős, mert a szindróma alapú dekódolás nem célszerű a konvolúciós kódokra.
A mátrix-szorzás szemléletessége miatt jól látszik a bemutatott példákban is, hogy a kódsorozat kiszámítása során, amint a transzponált üzenetvektort léptetjük jobbra, a kezdeti „bevezető rész" után, ugyanazokkal az elemekkel szorzódik az üzenetblokk más és más része. Mintegy végigcsúsznak ezek a szorzótényezők az üzenetsorozaton. Valamint ezt megfigyelve, az is szembeszökik, hogy mennyire „pazarló" a mátrix-leírás, hiszen csak egy keskeny sáv vesz részt a számításban. Valójában önként adódik a gondolat, hogy a polinomiális reprezentáció nagyon előnyös lehet, mert ott a nulla együtthatókat le sem kell írni.

Konvolúciós kódolás leírása polinomokkal

A könnyű követhetőség kedvéért folytassuk a fenti példákat! A (3,1)-es kód példájában, a generátormátrix közepén szereplő négyszögben három oszlopvektort látunk, amelyek egy-egy kódszegmens szimbólumait képezik. Ez úgy történik, hogy az éppen aktuális üzenetszimbólum és a két előző, az oszlopvektorral szorozva képezik a kódszimbólumokat. Írjuk le ugyanezt polinomokkal! A három kódszimbólum képzéséhez három generátorpolinom kell. Az első oszlopban egy 1-es van az alsó pozícióban, legyen a $g_1(x) = 1$! A második oszlop 1 0 1, amit írjon le a $g_2(x) = x^2 + 1$ polinom, illetve a harmadik

a $g_3(x) = x^2 + x + 1$ polinommal reprezentálható. A példa üzenetblokkját pedig a következő polinommal írhatjuk le:

$$u(x) = u_1 + u_3 \cdot x^2 + u_4 \cdot x^3 + u_5 \cdot x^4 \cdots = 1 + 1 \cdot x^2 + 1 \cdot x^3 + 1 \cdot x^4 \cdots$$

Elvégezve a szorzást, a következőt kapjuk:

$$c_1(x) = u(x) \cdot g_1(x) = (u_1 + u_3 \cdot x^2 + u_4 \cdot x^3 + u_5 \cdot x^4 \cdots) \cdot 1$$

$$c_2(x) = u(x) \cdot g_2(x) = (u_1 + u_3 \cdot x^2 + u_4 \cdot x^3 + u_5 \cdot x^4 \cdots) \cdot (x^2 + 1) =$$

$$= u_1 + (u_1 + u_3) \cdot x^2 + u_4 \cdot x^3 + (u_3 + u_5) \cdot x^4 \cdots$$

$$c_3(x) = u(x) \cdot g_3(x) = (u_1 + u_3 \cdot x^2 + u_4 \cdot x^3 + u_5 \cdot x^4 \cdots) \cdot (x^2 + x + 1) =$$

$$= u_1 + u_1 \cdot x + (u_1 + u_3) \cdot x^2 + (u_3 + u_4) \cdot x^3 + (u_3 + u_4 + u_5) \cdot x^4 \cdots$$

A kódszimbólumokat a polinomegyütthatók írják le, a közöttük végzendő műveletek GF(2)-ben végzendők. Az eredmény:

1 0 1 1 1

1 0 0 1 0

1 1 0 0 1

ami természetesen megegyezik a korábbival.

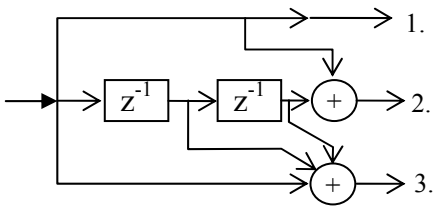
A generátorpolinomokat mátrixba is rendezhetjük, és a polinomos leírásban megadhatjuk a generátorpolinom-mátrixot, ami a példánkban:

$$\mathbf{G}^{(3,1)}(x) = \begin{bmatrix} g_1(x) & g_2(x) & g_3(x) \end{bmatrix} = \begin{bmatrix} 1 & (x^2 + 1) & (x^2 + x + 1) \end{bmatrix}.$$

A másik példa is említést érdemel, mert abban üzenetszimbólum párok hozzák létre a kódszószegmenseket. A generátorpolinom-mátrixnak ekkor két sora lesz, amelynek elemeit az eddigi tapasztalatok birtokában kiolvashatjuk a mátrix-leírásnál megismert generátormátrixból:

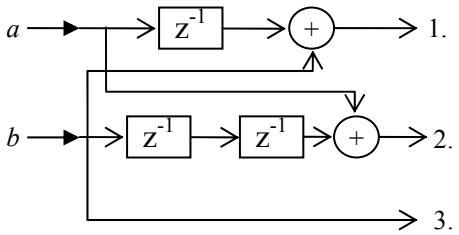
$$\mathbf{G}^{(3,2)}(x) = \begin{bmatrix} g_{11}(x) & g_{21}(x) & g_{31}(x) \\ g_{12}(x) & g_{22}(x) & g_{32}(x) \end{bmatrix} = \begin{bmatrix} x & 1 & 0 \\ 1 & x^2 & 1 \end{bmatrix}$$

A példát nem folytatjuk, mert sokkal fontosabb mondanivalónk is van a polinomokkal történő leírással kapcsolatban. A polinomok szorzása és osztása könnyen megvalósítható shift-regiszterekkel. Jelenleg a szorzást mutatjuk meg. Ha az $a(x)$ polinomot meg akarjuk szorozni a $b(x)$ polinommal, akkor kell egy shift-regiszter, amelyben a tárolók száma annyi, mint az egyik polinom fokszáma, és a megfelelő pontokon összeadást kell végezni. A megvalósítás szempontjából a legegyszerűbb a bináris eset, és a példáinkat úgy folytatjuk, hogy megszerkesztjük a fent meghatározott polinomokkal történő szorzást megvalósító shift-regisztereket. Az egyik polinom az üzenetszimbólumokat képviseli, amit a generátorpolinommal szorzunk. Az első példában 3 generátorpolinom van, a fokszámuk legfeljebb kettő. Kell tehát egy shift-regiszter két tárolóval. Mindhárom szorzást ugyanazzal a shift-regiszterrel végezhetjük el:



Az 1.-es jelű kimenetre közvetlenül a bemenő szimbólumok jutnak, erre a $g_1(x) = 1$ polinom érvényesül. A 2.-es jelű kimenetet $g_2(x) = x^2 + 1$ polinom határozza meg, azaz a kettővel

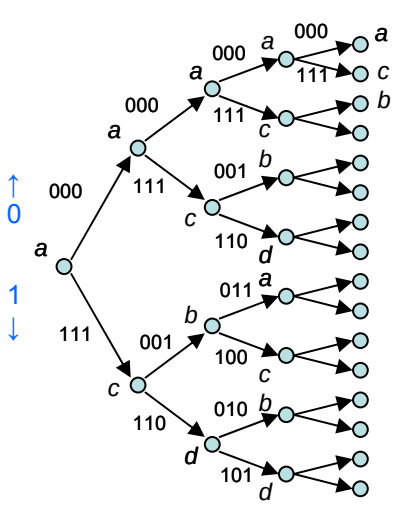
korábbi plusz (modulo 2) a jelenlegi, és a 3.-as jelű kimenetre $g_3(x) = x^2 + x + 1$ polinom dolgozik, azaz a kettővel korábbi, az eggyel korábbi és a jelenlegi modulo 2 összege.
Érdekes a másik példa is. Mivel itt $K = 2$, és $N = 3$, ezért a shift-regiszterekből 2 x 3-as készlet kell, amint azt a generátorpolinom-mátrix is mutatja. Viszont az is kiolvasható, hogy a felső sorhoz csak egy tároló elem kell, az alsóhoz pedig kettő, és nyilván itt is többszörösen kihasználjuk a shift-regisztereket:



Ellenőrizhetjük, hogy az *a* bemenetről az 1.-es kimenetre 1 késleltetéssel jutunk, azaz a polinom *x*, a 2.-es kimenetre közvetlenül, ami azt jelenti, hogy a polinom 1, és a 3.-as kimenetre nincs *a*-ról csatlakozás, tehát a polinom 0. A *b* bemenetről pedig az 1, x^2 és 1 polinomokat valósítottuk meg.
Az eddigi ismeretek birtokában nagyon fontos észrevenni, hogy bár adtunk kétféle leírási módszert a kódolásra, és az utóbbiból eljutottunk egy megvalósítási lehetőséghez, tehát praktikus kódolót is bemutattunk, nem adtunk semmilyen konstrukciós szabályt a generátormátrixra, vagy a polinomokra. Ennek pedig az az oka, hogy általában nem ismertek ilyen konstrukciós szabályok. A legtöbb ismert, és széleskörűen használt konvolúciós kódot számítógépes kereséssel találták meg, és táblázatokban foglalták össze a paramétereiket. Az egyik használt módszer a kód paramétereinek megadására a polinomegyütthatók oktális számokkal történő leírása. Így például a fenti (3,1)-es kódot az (1, 5, 7) polinomok határozzák meg, míg a (3,2)-eset ((1, 2, 0), (1, 4, 1)) polinomok.
A másik fontos észrevételünk pedig az kell legyen, hogy eddig még említést sem tettünk a konvolúciós kódok dekódolásáról. A dekódolási lehetőségek megértésében nagy segítséget jelent a kód, illetve a kódolás trellis-ekkel történő leírása.

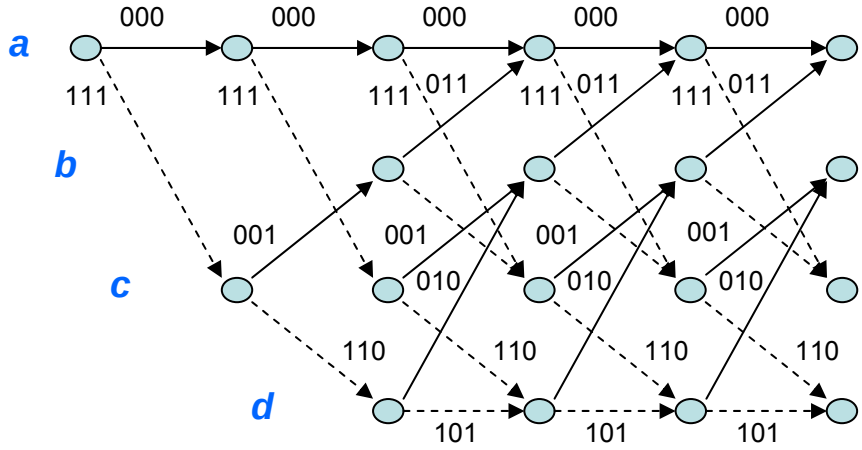
Konvolúciós kódolás leírása trellis-ekkel

A rácsalakú irányított gráfok, azaz a trellis-ek használata ugyan szokatlan lehet a kódolásról szerzett eddigi ismereteink alapján, viszont nagyon hasznosnak fog bizonyulni. Ezek a gráfok nem csak azt tudják ábrázolni, hogy egy adott kódparaméterekkel végzett kódolás során mely kódszószegmens keletkezik valamely üzenetszegmens hatására, hanem tömören megjelenítik az „egész történetet", tehát azt, hogy milyen „előélete" volt a kódnak, a kódolónak, amikor az éppen aktuális szegmens megérkezett, és azon végrehajtotta az előírt műveleteket.



Az ismerkedés a legegyszerűbb lesz egy példa keretében. Szerkesszük meg a már ismert (3,1)-es kód trellis-ét! Első lépésként rajzoljuk meg a mellékelt fa-gráfot! Tételezzük fel, hogy a shift-regiszter nullákkal van feltöltve, és nevezzük ezt az állapotot *a*-nak! A megérkező üzenetszimbólumok hatására a kódoló létrehoz egy kódot: vagy 000, ha 0 szimbólum érkezett, vagy 111, ha 1-es szimbólum jött. A beérkezett szimbólum hatására a kódoló egy új állapotba kerül, amelyeket a gráf csomópontjaihoz írt kisbetűk jelölnek, és az *a*-ból átmegy *c*-be az 1-es hatására, vagy megmarad *a*-ban, a 0 szimbólum hatására. A *c* jelű

állapotban az első tároló bemenetén 1-es, a másodikban 0 van. Ha a beérkező 1-es hatására a *c* jelű állapotba jutott a kódoló, akkor a következő 0 üzenetszimbólum a *b* jelű állapotba, míg az 1-es a *d* jelű állapotba viszi a kódolót, miközben az a 001, illetve az 110 kódszegmenseket generálja. Végül már csak az maradt, leírjuk mi történik, ha shift-regiszter a *b* jelű, vagy a *d* jelű állapotában van, és új üzenetszimbólum érkezik. Könnyen ellenőrizhetően a *b* jelű állapotból az *a* jelű, vagy a *c* jelű állapotba kerül a 0, illetve az 1-es hatására, miközben 011, illetve 100 szegmenst generál, míg a *d* jelű állapotból a *b* jelű, vagy a *d* jelű állapotba kerül a 0, illetve az 1-es hatására, miközben 010, illetve 101 szegmenst generál.
A fa-gráftól csak egy lépés a trellis, mert csupán azt kell észrevenni, hogy felesleges a fa gráf korlátlan szétterülését, terebélyesedését megengedni, hiszen észrevehető hogy ismétlődések vannak benne. Ha elhagyjuk a felesleges ismétlődéseket illetve elvégezzük a lehetséges összevonásokat, akkor a trellis-t kapjuk:



A trellis csomópontjait, amelyek a shift-regiszter állapotát jelölik, sorokba rendeztük, és egy-egy sorban azonos állapotok találhatók. Így kialakul a rács, amit még azzal színeztünk, hogy folytonos vonalat használtunk az élre, ha 0 üzenet-szimbólum hozza létre, illetve szaggatott vonallal jelöltük az 1-esek hatására létrejövő éleket.
A trellis éppen úgy meghatározza a kódot, mint a generátormátrix, vagy a generátorpolinom-mátrix, viszont van néhány előnyös tulajdonsága is, amelyek a kód teljesítőképességének a feltárásában, illetve a dekódolási algoritmusok megértésében játszanak szerepet. Azonban van még egy hasznos leírási lehetőség a konvolúciós kódra, ami szinte nélkülözhetetlen a mennyiségi viszonyok meghatározásánál, de ezt csak a dekódolás megismerése után mutatjuk meg.

Konvolúciós kódok dekódolása

Várakozásunk szerint a konvolúciós kódolás előnyösebb lehet, mint a blokk kódolás, mert nem szakad meg a szimbólumok között kialakuló kapcsolatrendszer a blokkhatárokon. Ez a várakozásunk teljesülni fog, az elérhető eredményeket a dekódolás megismerése után mutatjuk meg. Jelenleg azonban a fenti előnyt eredményező problémával kell szembenéznünk, azaz miként állapítsuk meg egy vett sorozatról, amely akár végtelen hosszú is lehet, hogy legvalószínűbben melyik küldött sorozatból jöhetett létre. Először idézzük fel, hogy miként jártunk el a blokk kódolásnál!
Feltételeztük, hogy ismerjük blokkok határait, és egy vett blokkról, amelyet a csatornában észlelt jelből a demodulátor előállított, megállapítottuk, hogy melyik kódszóra hasonlít leginkább, és az annak megfelelő üzenetet fogadtuk el, mint a legvalószínűbbet. A hasonlóságot a Hamming távolsággal mértük, és így a legkisebb Hamming távolságú sorozatot fogadtuk el. A feladat elvégzéséhez vektor- vagy polinom-algebrai módszereket alkalmaztunk.

Konvolúciós kódok esetén a kódsorozat mérete (hossza) nem kötött. Elvileg képezhetnénk blokkokat, de ez nem igazán oldaná meg a dekódolási problémát, viszont rontaná a kód hatékonyságát: felesleges szimbólumokat kellene beiktatni, és a blokkhatárokon megszakadnának a kapcsolatok, tehát valójában blokk kódot képeznénk.
A dekódolás alapelvét, tehát azt, hogy egy vett sorozat észlelésekor a legvalószínűbb küldött sorozatot kell elfogadni, nyilván meg kell őrizni, de a mátrix- vagy polinom-algebrai módszerekről valószínűleg le kell mondani, mert a végtelen méret ezek használatát praktikusan lehetetlenné teszi.
Sajnos még egy dologról le kell mondanunk, nevezetesen a maximum a-posteriori valószínűségen alapuló (MAP) dekódolásról, de ez nem nagyon fájó. Ugyanis nem célszerű olyan modellt konstruálni, amely ismertnek tételei fel a küldött sorozatok a-priori valószínűség-eloszlását, így megelégszünk a maximum likelihood (ML) döntéssel, ami gyakorlatilag azt jelenti, hogy valamennyi sorozatot azonos valószínűségűnek tételezünk fel.
Ennek értelmében meg kell állapítanunk, hogy a küldött sorozatok milyen valószínűség-eloszlással hozzák létre a csatorna kimenetén vehető sorozatokat. Természetesen ezt valamilyen csatorna-modell esetén lehet meghatározni. Az egyszerűség kedvéért legyen a vizsgált csatorna BSC, amelyet a p bit-hibavalószínűsége egyértelműen meghatároz. Ekkor könnyű kiszámítani (blokk kódoknál sokszor használtuk) az alábbi valószínűséget:

$$\mathbf{P}(d \text{ számú hiba } n\text{-ben}) = \binom{n}{d} \cdot p^d \cdot (1-p)^{n-d},$$

illetve, a most érdekes feltételes valószínűséget:

$$\mathbf{P}(\textit{vett sorozat} \mid \textit{küldött sorozat}) = p^d \cdot (1-p)^{n-d},$$

ahol d a *küldött sorozat* és a *vett sorozat* Hamming távolsága (azaz d bitben térnek el egymástól az n hosszúságú sorozatok).
Ezután vegyük elő a kódot leíró trellis-t! A trellis ágaihoz kódsorozat-szegmensek vannak rendelve. Ha a fenti valószínűségben feltételezzük az egyes ágak kódszegmenseit, akkor kiszámíthatjuk a vett sorozat egy-egy részének valószínűségét. Vegyük ezeknek a valószínűségeknek a logaritmusait, és használjuk a hasonlóság mértékeként. A trellis egymást követő ágai egy-egy utat képeznek, amelyekre vonatkozóan a fenti mértékeket összeadhatjuk (a valószínűségeik összeszorzódnak). Véges hosszúságú utaknál maradva, amelyek r ágból tevődnek össze, annak a valószínűségét, hogy egy *küldött sorozat* esetén valamely út mennyire valószínű egy adott tulajdonságú csatornán, az alábbi kifejezéssel adhatjuk meg:

$$\textit{egy út mértéke} := \mu = \sum_{i=1}^{r} \log\left[p^{d_i} \cdot (1-p)^{n-d_i}\right].$$

Érdemes elvégezni a következő egyszerű átalakítást:

$$\mathbf{P}(\textit{vett sorozat} \mid \textit{küldött sorozat}) = \left(\frac{p}{1-p}\right)^d \cdot (1-p)^n,$$

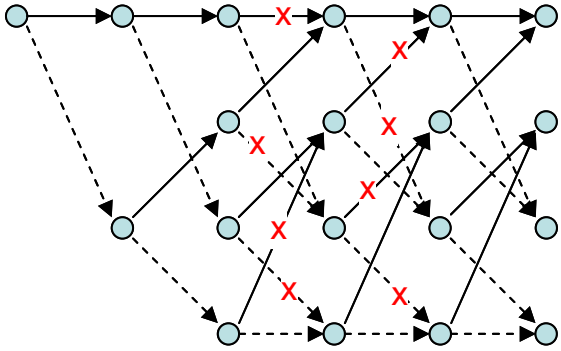
így az egyes ágak mértéke az alábbi lesz:

$$\mu_i = d_i \cdot \log\left(\frac{p}{1-p}\right) + n \cdot \log(1-p),$$

ami jól mutatja, hogy ½-nél kisebb bit-hibavalószínűségű csatorna esetén mindkét logaritmus argumentuma 1-nél kisebb lévén, értékeik negatívak, tehát a legkisebb negatív mértékű úté lesz a legnagyobb mérték. Végül levonhatjuk a következtetést, hogy a trellis-ben egy-egy út valószínűségének a mértékéül használhatjuk a Hamming távolságot, de nem a legnagyobb, hanem a legkisebb Hamming távolságú út lesz a legvalószínűbb (összhangban a fentiekkel és a várakozásunkkal). A fentiek alapján működő dekódolási módszert Viterbi dolgozta ki.

A Viterbi dekódolás

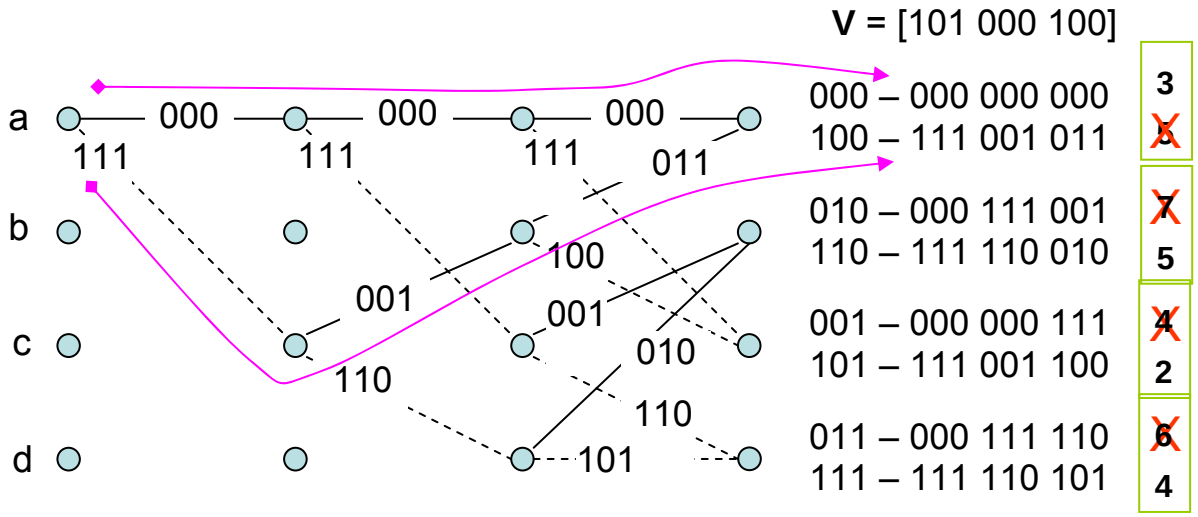
Viterbi észrevette, hogy egy vett sorozat esetén a sorozatot legvalószínűbben eredményező útnak a trellis-en történő megkereséséhez nem kell az összes lehetséges út mértékét meghatározni, mert ki lehet használni bizonyos kötöttségeket. A trellis-en az ágak csomópontokból indulnak, és csomópontokba futnak össze. Egy-egy csomópontba összefutó ágak megszerzett mértékéhez adódik hozzá a következő ág mértéke. A csomópontban az ágak nem keresztezik egymást, hanem egyesülnek, így megőrizni csak a nagyobb mértéket érdemes, és azt az ágat (útrészt), amelyik kisebb mértékkel érkezett, ki lehet ejteni a versenyből. Így minden csomópontnál csak egy *túlélő* utat kell megőrizni. A mellékelt ábrán illusztráljuk a megőrzendő túlélőket. Bináris esetben a csomópontba érkező kettő közül a kisebb mértékűt elfelejthetjük. Ezeket át-ixeltük.



Emlékeztetünk, hogy a feladat annak az útnak a megkeresése, amelyre fennáll:

$$\max \sum_{i=1}^{r} \log\left[p^{d_i} \cdot (1-p)^{n-d_i}\right], \text{ vagy } \min \sum_{i=1}^{r} d_i \ .$$

A feladat elvégzését, azaz a Viterbi dekódolást a már többször használt, egyszerű példánkon, a (3,1)-es kódon mutatjuk be. A megfelelően felcímkézett trellis a következő:



A példa kapcsán hangsúlyoznunk kell, hogy nem megy az általánosság rovására, ha egy csupa nulla sorozatot tételezünk fel *küldött sorozat*-ként, mert a kód lineáris. Így elindulva az a – állapotból, megvizsgáljuk, hogy a kódoló kényszerhosszán mekkora a Hamming-távolság a trellis egyes, záródó útjain a vett sorozat és a záródó utak sorozatai között! Minden záródó útpár közül a kisebb Hamming-távolságút őrizzük meg, és megjegyezzük a csomópontra akkumulált Hamming-távolság értékét! Az ábrán a **v** – jelű vektort tételeztük fel vett sorozatként, és összeszámoltuk az egyes, 3 ág hosszúságú utak Hamming távolságát a **v** sorozattól. A találkozási pontokban feltüntettük a kapott értékéket, és a nagyobbat áthúztuk. Ezután egy lépéssel (ággal) továbblépünk, ismét kiszámoljuk a Hamming távolságokat, hozzáadjuk az előzőleg megőrzöttekhez, és megint a kisebbeket őrizzük meg. Ezt így

folytatjuk az üzenet végéig. Amikor véget ér az üzenet, akkor a legnagyobb mértékű, tehát legkisebb Hamming távolságú csomópontot fogadjuk el, mint a legvalószínűbb út végét. Ekkor ebből a csomópontból, a lépésenként megőrzött ágakon visszafelé lépkedve kiolvashatjuk a legvalószínűbb küldött kódsorozatot, illetve a legvalószínűbb üzenetet. Ha a legvalószínűbb küldött kódszót összevetjük a vett sorozattal, akkor megállapíthatjuk a becsült hibákat.
Figyelmesen követve a tennivalókat, szerepel egy kitétel, amely nem minden felhasználás esetén teszi lehetővé az ismertetett módszer alkalmazását, nevezetesen: „folytatjuk az üzenet végéig". Amennyiben tényleg ezt tennénk, akkor olyan késleltetés keletkezhetne, amely bizonyos esetekben elfoghatatlan lenne. Azonban ezen is túl lehet lépni.

A módosított Viterbi algoritmus

Az elfogadható mértékű dekódolási késleltetés érdekében – vállalva az esetleges tévedést is – nem várjuk végig az üzenetet, hanem egy általunk választott hosszúságú kódsorozat beérkezése és kiértékelése után megkezdjük a „visszatekintést" az addig legvalószínűbb üzenetszimbólumokra, és ettől kezdve, mindegyik kódszószegmens kiértékelése után egy-egy újabb, feltételezett üzenetszimbólumra adunk becslést. A csatornában lévő jel-zaj viszonytól függően hosszabb-rövidebb sorozat dekódolása után már kicsi a valószínűsége, hogy az utak mértékeinek sorrendjében jelentős változás következzék be, így ez a módszer közel optimális dekódolást tesz lehetővé.

A lágy döntéses Viterbi dekódolás

A Viterbi dekódolásról eddig elmondottakban azt tételeztük fel, hogy a demodulátorban „kemény" döntés születik a beérkező szimbólumra, és a dekóder egy szimbólumsorozatot kap. Pedig számos esetben ezt a döntést nem lehet nagy biztonsággal elvégezni, és így hibás szimbólum jön létre. De hát azért van a dekóder, hogy kijavítsa a hibát! Viszont könnyebb dolga lenne, illetve biztonságosabban, nagyobb valószínűséggel adhatna a dekóder jó eredményt, ha figyelembe vehetné azt a bizonytalanságot is, amellyel a döntőkészülék szembesült. Tehát ahelyett, hogy – esetleg tévesen – eldöntött szimbólumokat adnánk a dekódernek, adjuk oda neki a demodulátor által tapasztalt, a csatornából érkező értékeket, amelynek alapján a döntőkészüléknek kellene működnie.
Egy kicsit pontosabban, és a megvalósíthatóságra is tekintettel, a demodulátor például egy bináris átvitelnél nem azt dönti el, hogy 1-es, vagy 0-s szimbólum jött, hanem, hogy a feldolgozott zajos jel a kijelölt értéktartományok közül melyikbe esik bele. Ezért joggal mondhatjuk hogy dönteni is kell, de ugyanakkor a „kemény" bináris döntést „meglágyítottuk".
A lényeges különbséget a kemény döntéses dekódoláshoz képest a trellis ágai, illetve útjai mértékének a meghatározása jelenti. Tételezzük fel, hogy a csatornában additív, gaussi, fehér zaj zavarja az átvitelt! Ekkor a demodulátor kimenetén egy normális eloszlású valószínűségi változó képezi a döntés alapját. Ha ennek eloszlását σ szórás és $\pm A$ várhatóérték jellemzi, akkor annak a valószínűsége, hogy az *i*-edik lépésben az (*r*) jelű ág *j*-edik időrésében *v* értékű lesz, feltéve, hogy a küldött szimbólum *c* volt, a következő:

$$P(v_{ij} \mid c_{ij}^{(r)}) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{[v_{ij} - A(2 \cdot c_{ij}^{(r)} - 1)]^2}{2\sigma^2}}$$

(a fenti képletben $c_{ij}^{(r)}$ egy 0 vagy 1 értékű bináris változó).

Ha egy-egy ág, azaz kódszegmens *n* szimbólumból áll, akkor az *i*-edik lépésben a mérték:

$$\mu_i^{(r)} = \sum_{j=1}^{n} \log P(v_{ij} \mid c_{ij}^{(r)}) .$$

Képezve a logaritmust, a konstans tagot elhagyhatjuk, mert a mértékeket csak egymáshoz kell viszonyítani, és így az alábbi összeget kapjuk:

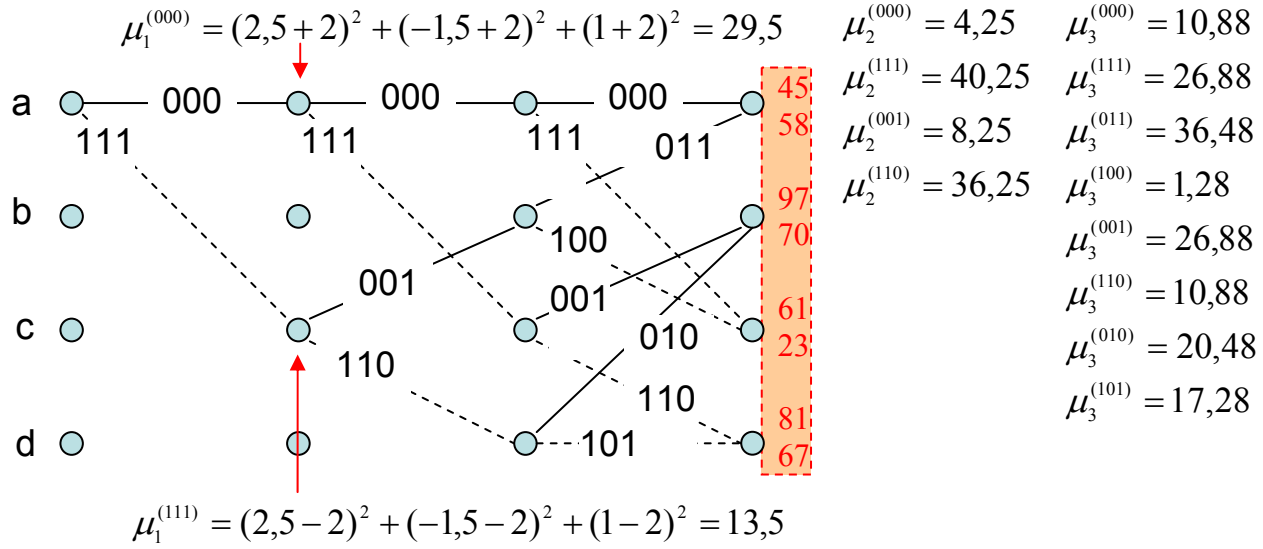
$$\mu_i^{(r)} = \sum_{j=1}^{n} \left[v_{ij} - A(2 \cdot c_{ij}^{(r)} - 1) \right]^2$$

A dekódolást hasonlóan végezzük, mint eddig, csak most nem a Hamming távolságokat használjuk, hanem a fenti módon számított mértéket. Jelenleg a kisebb mérték a jobb, mert az jelenti a kisebb euklédeszi távolságot.

Az alábbiakban illusztráljuk a lágy döntést a korábbi példa átalakításával. Jelenleg tehát a demodulátorból nem a **V = [101 000 100]** bináris vektor érkezik, hanem például a következő:

V = [2,5 -1,5 1 -1 -3 -0,5 1,2 -1,2 -2] ,

azaz végezze a demodulátor a lágy döntést egy tizedes pontossággal, és legyen $A = 2$. (A gyakorlatban persze nem tizes számrendszerben működik demodulátor, hanem néhány bináris jegy használatos.) A trellis, és a számított mértékek a következők:

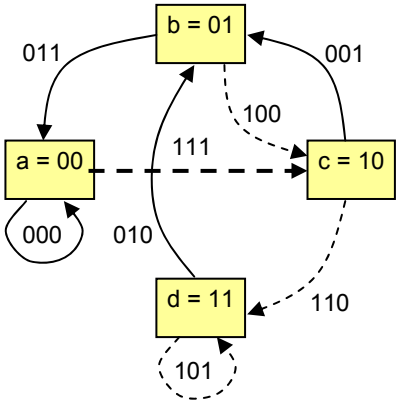


A túlélő utak a 45, a 70, 23 és a 67 mértékűek (kerekítettünk). Ha máris dönteni akarnánk, akkor nyilván a 23-as mértékűt választanánk, ami az 111 001 100 sorozatot tételezi fel.

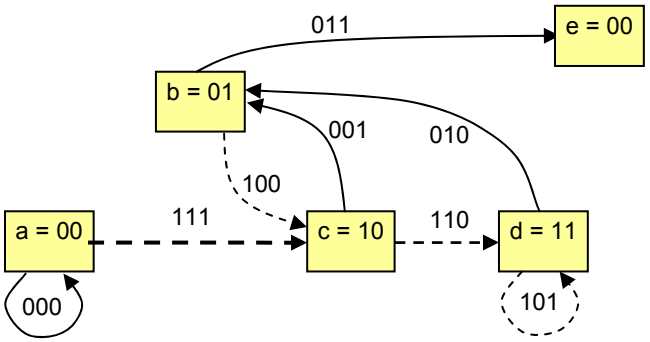
A konvolúciós kódolásról eddig elmondottak alapján tudjuk, hogy milyen módon írhatjuk le a kódolás folyamatát, hogyan kell a kódot generálni, és vett sorozatokat dekódolni, de a konvolúciós kód teljesítőképességének a meghatározásához, még egy további leírási módszerrel kell megismerkednünk.

Konvolúciós kódolás leírása állapot-átmenet diagrammal

Mivel a kódoló egy véges állapotú automatával valósítható meg, a kódolás legtömörebb leírására az *állapot-átmenet diagramm-ot* használhatjuk. Ismét példán keresztül szeretnénk a viszonyokat megvilágítani, ezért tekintsük a már jól ismert (3,1)-es kódunkat. A kódolónak négy állapota van, amelyeket az abc első négy betűjével jelöltünk, és a jelenlegi diagrammban a négy állapotot jelképező dobozokba írtunk. A dobozok közötti, vagy néhol visszatérő utak képviselik az állapot-átmeneteket, amelyek során azok a kódszavak képződnek, amiket mellékük írtunk. Az átmeneteket itt is megkülönböztettük attól függően, hogy 0-s üzenetszimbólum (folytonos vonal), vagy 1-es szimbólum (szaggatott vonal) váltja-e ki.



Ez a diagramm azonban csak egy újabb ábrázolás lenne, ha nem tudnánk belőle a kód egyik legalapvetőbb teljesítmény-jellemzőjét, a kódszótávolságot meghatározni. Persze ez nem egyszerű feladat, hiszen a keletkező kódszavak hossza nem kötött, sőt akár végtelen is lehet. A lineáris blokk kódok esetén ez egy sokkal egyszerűbb feladat volt.
Az elvégzendő feladat érdekében alakítsuk át kicsit az állapot-átmenet diagrammot, oly módon, hogy a kezdő (00) állapot megismétlésével hozzunk létre egy ú.n. végállapotot, azaz nevezzük végállapotnak azt, amikor a 00 kezdetről ismét elérjük 00 állapotot. Ezt a diagramot szokták *állapot-folyam-gráf*-nak is nevezni:



A könnyű összehasonlíthatóság kedvéért csak azokat az elemeket mozdítottuk el, amelyek az áttekinthető ábrához voltak szükségesek, és elhelyeztük az e jelű végállapotot.
A blokk-kódolásból tudjuk, hogy egy kód hibakezelő képessége a kódszavak távolságától (Hamming), pontosabban a két legközelebbi kódszó távolságától, a *kódtávolságtól* függ. Konvolúciós kódok esetén nincsenek meghatározott hosszúságú kódszavak (ez az egyik előnyük), hanem tetszésszerinti hosszúságú, elvileg akár végtelenig tartó kódsorozatokról beszélhetünk. Ezeknek a kódsorozatoknak a csatornában keletkező hibák ellenére történő megkülönböztethetőségét a közöttük lévő távolságok határozzák meg. A távolságot legegyszerűbben Hamming értelemben vehetjük, amikor azt tételezzük fel, hogy a dekódolóra egy – a kód-ábécé-nek mefelelő – szimbólumsorozat érkezik a demodulátort követő, vagy annak részeként felfogható döntőkészülékből. Ezt nevezzük kemény döntési eljárással kombinált dekódolásnak. (A blokk-kódoknál csak erről beszéltünk. Konvolúciós kódoknál gazdaságosan realizálható az ú.n. lágy-döntés is.)
Könnyű belátni, hogy a kódsorozatok közötti távolságot befolyásolja a hosszuk is. Ezért ú.n. *távolság-profil*-t határozhatunk meg egy kódra, amely azt mutatja meg, hogy különböző feltételek mellett milyen lesz a sorozatok távolságának az eloszlása, azaz mekkora távolságú sorozatok, mekkora számban fordulnak elő. Ez a távolság-eloszlás majd lehetőséget teremt a hibakezelő képesség jellemzésére.

Konvolúciós kód távolság profilja

A távolság profil meghatározása a kód *állapot-folyam-gráf*-jából történhet.
Az *állapot-folyam-gráf* útjai címkézettek, egyrészt az úthoz tartozó kódok, másrészt az üzenet-szimbólumok képviselik a címkéket. Azt, hogy milyen utakon juthatunk az a = 00 állapotból az e = 00 állapotba, tehát milyen „kitérőket”, hurkokat írhatunk le, az alábbi módon határozhatjuk meg. Egyenleteket írhatunk fel az egyes állapotok közötti átmenetekre. Ezekből az egyenletekből kifejezhetjük az a -ból e -be történő lehetséges átmenetek jellemzőit, amit transzfer függvénynek is neveznek, és itt az alábbi módon jelölünk:

$$\left.\begin{aligned} c &= M \cdot B \cdot D^3 \cdot a + M \cdot B \cdot D \cdot b \\ d &= M \cdot B \cdot D^2 \cdot c + M \cdot B \cdot D^2 \cdot d \\ b &= B \cdot D \cdot c + B \cdot D \cdot d \\ e &= B \cdot D^2 \cdot b \end{aligned}\right\} \quad \frac{e}{a} = T(B, M, D)$$

Az M kitevője azt jelöli, hogy az adott átmenetet melyik üzenet-szimbólum váltja ki (0 vagy 1), a B kitevője jelöli az átmenetben szereplő ágak (branch) vagy élek számát (természetesen a diagramban az átmenetek mindig egy-egy élen történnek, de a trellis mentén már nem ez a helyzet). Végül a legfontosabb, a D kitevője jelöli az átmenetnek a csupa 0 kódtól való Hamming távolságát. Például az első egyenlet szavakban azt jelenti, hogy a c állapotba eljuthatunk a –ból egy 1-es hatására (M) egy lépésben (B), és közben 3 Hamming távolságú (D^3) a kódszó a 000-tól mérve.
A négy egyenletet a jobboldalon jelzett hányadosra akarjuk megoldani. Az eredmény:

$$\frac{e}{a} = T(B,M,D) = \frac{B^3 \cdot M \cdot D^6}{1 - B \cdot M \cdot D^2(1+B)}$$

Vegyük észre, hogy egy mértani sor összege a kapott kifejezés. Ez egy végtelen összeg, amelynek egyes tagjai megmondják, hogy milyen utakon lehet a trellis-ben a 0 állapotból kilépve a 0 állapotba visszajutni. Felírva az összeg első néhány tagját:

$$T(B,M,D) = B^3 \cdot M \cdot D^6 + B^4 \cdot M^2 \cdot D^8 + B^5 \cdot M^2 \cdot D^8 +$$
$$+ B^5 \cdot M^3 \cdot D^{10} + 2 \cdot B^6 \cdot M^3 \cdot D^{10} + \cdots$$

kiolvashatjuk, hogy legrövidebben 3 ágon (lépésben), egy 1-es hatására (ezért léptünk ki a 0 állapotból), egy 6 súlyú úton jutunk vissza a 0 állapotba. A 6 súlyú útnak természetesen 6 a Hamming távolsága a 0 súlyú úttól. A következő „hurok” 4 lépésben, egy 8 távolságú úton, két 1-es szimbólum hatására következik be. És így tovább.
De miért érdekel bennünket egyáltalán a kódnak ez a tulajdonsága? Mert ebből tudunk következtetni a hibakezelő képességére. Ugyanis az általánosság elvesztése nélkül feltételezhetjük (lévén a kód lineáris), hogy a forrás a csupa 0 üzenetszimbólumot bocsátja ki, és így a csatornára a csupa 0 szimbólumsorozat kerül. Amennyiben téves döntés következtében a vett szimbólumsorozat ettől eltér, akkor a dekódolás során a legvalószínűbbé válhat egy a csupa 0-ástól eltérő út. A kérdés az, hogy ez milyen arányú döntési tévesztésnél következik be.

Konvolúciós kód hibavalószínűsége

Egyébként, ha nem vagyunk kíváncsiak a hurokágban lévő üzenetbitek számára (U), vagy a hurkok hosszára (B), akkor a megfelelőt a folyamat gráfban 1-el helyettesítve *T(B,D),* illetve *T(D)* is megkapható. Ha a szereplő ágak számától akarjuk függetleníteni magunkat, akkor B –t 1-el kell helyettesíteni, és így a következő eredményt kapjuk:

$$T(M,D) = M \cdot D^6 + 2 \cdot M^2 \cdot D^8 + 4 \cdot M^3 \cdot D^{10} + 8 \cdot M^4 \cdot D^{12} + \cdots$$

Összeg alakjában is felírhatjuk az eredményt:

$$T(M,D) = \sum_{d=6}^{\infty} \alpha_d \cdot M^{(d-4)/2} \cdot D^d \ ,$$

ahol α_d adja meg a d hosszúságú hurkok darabszámát , ami a példában:

$$\alpha_d = \begin{cases} 2^{(d-6)/2}, & \text{ha } d \text{ páros} \\ 0, & \text{egyébként} \end{cases}$$

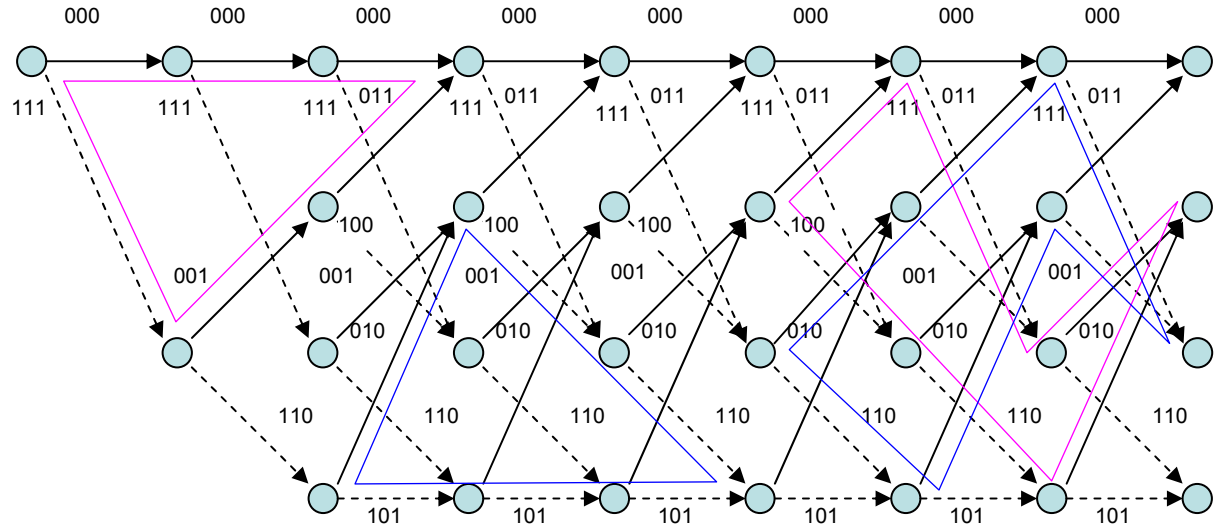
Megkaphatjuk a hamis ágon keletkező hibás bináris szimbólumok számát is *T(D,M)* –ből, ha deriváljuk M szerint, és a derivált értékét vesszük M = 1-nél. (Pontosabban ez nem explicit módon adja meg a hibák számát, hanem az szorzótényezőként szerepel)

$$\left.\frac{\partial T(M,D)}{\partial M}\right|_{M=1} = \sum_{d=6}^{\infty} \alpha_d \cdot \frac{d-4}{2} \cdot D^d$$

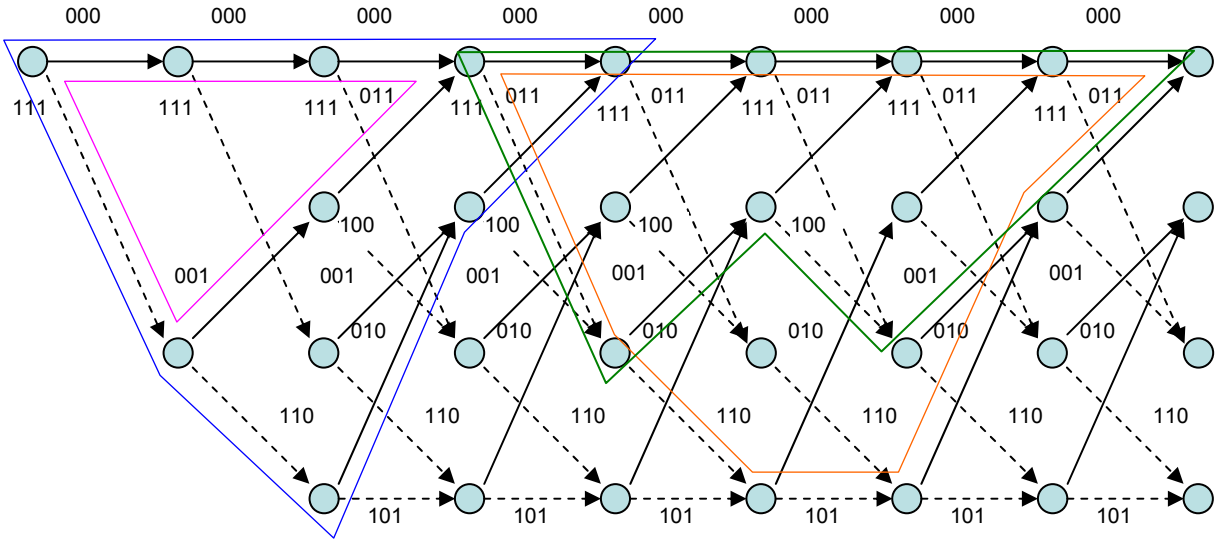
Az eddigieknek az a jelentősége, hogy az így kapott összefüggéssel egy felső korlátot adhatunk a bit-hibavalószínűségre, amit már most leírunk, de megértéséhez még egy kis vizsgálódásra van szükség:

$$P_e < \left.\frac{\partial T(M,D)}{\partial M}\right|_{M=1, D=\sqrt{4p(1-p)}}$$

Vizsgáljuk meg a dekódoló tévedésének, azaz a hibás üzenetszimbólum keletkezésének a mechanizmusát! A kód linearitása miatt bármely sorozatot feltételezhetjük a vizsgálathoz, legyen ez például a csupa nulla sorozat. Könnyű ellenőrizni, hogy az eddigi példáinkban is gyakran szereplő kódra a legrövidebb hurkok, amelyek 3 lépés hosszúak, a különböző sorozatokra csak eltérő alakúak, de megegyeznek a paramétereik, nevezetesen az első lépésben 3, a másodikban egy és a harmadikban 2 kódszimbólum ez eltérés az ágak között. Ezt illusztráltuk az alábbi ábrán néhány 3 hosszúságú hurok megjelölésével:



Van azonban ennél sokkal fontosabb mondanivalónk is, de a zavarosság elkerülése érdekében megismételjük a trellis-t, és berajzolunk hosszabb hurkokat a 00 állapotból kiindulva, egy 3 lépés hosszú hurkot, egy 4 lépés hosszút, és két 5 lépéses hurkot (zöld és narancs):



A bejelölt hurkokat megvizsgálva, megállapíthatjuk, hogy adott számú csatornaszimbólumból mennyit kell tévesen értékelni ahhoz, hogy ne az igaz utat őrizzük meg, és így a dekódolásnál meghatározott mennyiségű üzenethibát generáljunk. (Ha a téves ág mértéke nagyobb, akkor azt őrizzük meg, és a döntésnél ezen megyünk vissza, bármi történik is a későbbiekben.)

Nézzük a legrövidebb hurkot! Ha a csatornába küldünk 9 darab 0 szimbólumot, akkor a hurok összefutó csomópontjában a téves (alsó) út elfogadásához az kell, hogy legalább 3 esetben 1-esre döntsünk, mert így megegyezik a helyes út (felső) és a téves út mértéke, ami sorsolásos esetben ½ valószínűségű tévedés, míg 4 vagy több 1-es detektálásakor biztosan a téves utat fogadjuk el. az adott hurokra vonatkozóan a téves út elfogadása a 3 üzenetszimbólumból egynek az eltévesztését eredményezi.
Amennyiben ugyanezt végigszámláljuk a többi bejelölt hurokra, akkor azt kapjuk, hogy a 4 lépéses (kék) esetben 4 téves döntés vezet ½ valószínűséggel a hamis út elfogadására, míg 5 vagy több hibás döntés esetén mindig a hamis utat fogadjuk el. Tehát 12 csatornaszimbólumból ennyinek az eltévesztése fog 2 üzenetszimbólumot eltéveszteni. Nem folytatjuk, mert jól látszik, hogy a kód teljesítőképességének a meghatározásához ezzel a számlálással nem jutunk közelebb, bár így megértjük a hibás dekódolás mechanizmusát.
Gondoljuk végig, hogy mi is történik a dekódolás egy-egy lépésében! A hozott mértékekhez hozzáadjuk az új ágakra kiszámított mértékeket, és minden csomópontra vonatkozóan, az ott összefutók közül a nagyobb mértékűt választjuk túlélőnek.
Mivel a csupa nulla sorozat küldését tételezzük fel, az egyedüli „igaz” út a felső. A hamis utak alul futnak. Az nem érdekes, hogy milyen módon jött létre a hamis út, egészen addig, ameddig a vizsgált csomópontnál, a 0 állapottal való találkozáskor nem válik túlélővé. Ha a hamis út válik túlélővé, akkor a dekódolásban egy hiba-esemény következik be. A hiba-esemény által okozott üzenethibák száma attól függ, hogy az éppen csatlakozott és túlélővé vált hamis út mikor vált el az igaz úttól. Ezeket a jellemzőket a távolság-profil kiszámításánál ismertetett módon meg lehet határozni egy adott kódra.
Maga a mechanizmus érthető, de az még nem látszik, hogy miként lehet a kód teljesítőképességét jellemezni.
Először határozzuk meg annak valószínűségét, hogy a kezdeti állapotból kiindulva egy-egy ilyen csomópontba összefutó utak esetén, a *túlélő* kiválasztásánál a hibás ágat választjuk!
Kemény döntés esetén a csatlakozó út-párból akkor választjuk a rosszat, ha annak a Hamming távolsága kisebb a vett sorozattól, mint a helyes úté. A hamis út választásának valószínűsége sajnos még attól is függ, hogy a csatlakozó hamis út távolsága az igaz úttól, páros vagy páratlan. Kemény döntést, bináris szimmetrikus csatornát feltételezve:

$$P_{\text{pár}}(d_{\text{ps}}) = \sum_{i=(d+1)/2}^{d} \binom{d}{i} \cdot p^i \cdot (1-p)^{d-i}$$

$$P_{\text{pár}}(d_{\text{ptln}}) = \sum_{i=d/2+1}^{d} \binom{d}{i} \cdot p^i \cdot (1-p)^{d-i} + \frac{1}{2} \cdot \binom{d}{d/2} \cdot p^{d/2} \cdot (1-p)^{d/2}$$

(ha a két út, amelyek egy csomópontban összefutnak, *d* Hamming távolságúak, akkor felénél több hiba kell a hamis út választásához).
Továbbá a csatlakozáshoz megérkező hamis ág több különböző hosszúságú út utolsó szakasza lehet, és bár ismerjük a lehetséges hurkok hosszát, nem ismert egyszerű zárt kifejezés a hamis túlélő választásának valószínűségére, amit *első hiba-esemény* valószínűségnek hívunk, angolul *first-event error* valószínűségnek szokták nevezni.
Megadhadjuk viszont az *első hiba-esemény* valószínűségének felső becslését. Ennek egy viszonylag egyszerű módja lehet, ha összeadjuk valamennyi olyan hamis út választásának valószínűségét, amelyek hosszabbak a *szabad úthossz*-nál, tehát egyáltalán szóba jöhetnek. Természetesen súlyozott összeget képezhetünk, hiszen a távolság-profilból ismerjük a különböző hosszúságú hurkok számát is. Az így kapható eredmény részben azért felső becslés, mert ezzel az összeggel implicit módon végtelen hosszú sorozatot tételezünk fel, részben pedig azért, mert a különböző hosszúságú utak nem különállóak, hanem menet közben összeolvadhatnak:

$$P_e < \sum_{d=d_f}^{\infty} \alpha_d \cdot P_{\text{pár}}(d),$$

ahol $P_{\text{pár}}(d)$ annak a valószínűsége, hogy egy vett bináris sorozat közelebb van a csupa nullától d távolságra lévő sorozathoz, mint a csupa nullához. Ezt a valószínűséget felülről becsülhetjük az itt nem részletezett módon kiszámítható, ú.n. Chernoff korláttal:

$$P_{\text{pár}}(d) < \left[4 \cdot p(1-p)\right]^{d/2}$$

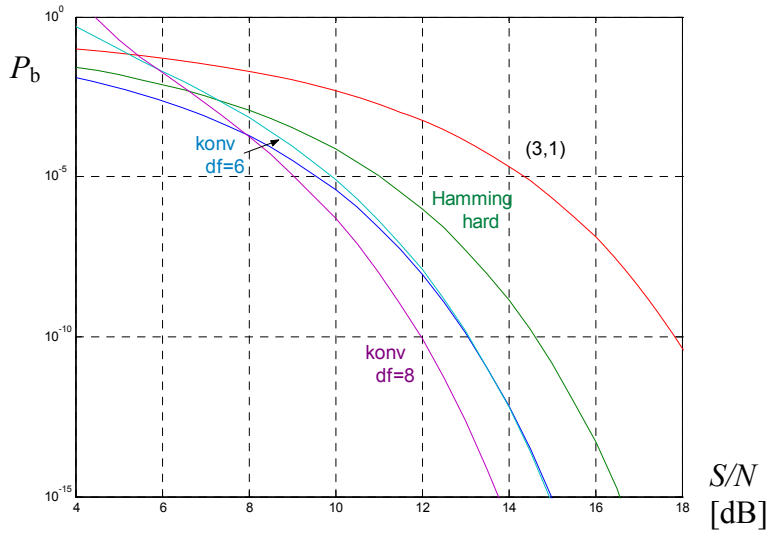
Ezt behelyettesítve az első hiba-esemény felső korlátjaként kapjuk:

$$P_e < \sum_{d=d_f}^{\infty} \alpha_d \cdot [4p(1-p)]^{d/2} = T(D)\Big|_{D=\sqrt{4p(1-p)}}$$

Ahol felhasználtuk a késleltetés profilra kapott kifejezést a rövid leírás kedvéért.
Még egy kis lépés, és meghatározhatjuk a dekódolt üzenetben a bit-hibavalószínűséget is (pontosabban persze egy felső korlátot arra). Ez azért egyszerű, mert a távolság profil meghatározásakor bevontuk az egyes átmenetekhez tartozó üzenet-1-esek számát is. Ezt jelölte M hatványkitevője.
Most semmi mást nem kell tenni, mint megállapítani, hogy hány üzenet-bit hibát eredményeznek a tévesen választott hurkok. Ezt jelöltük bétával. Ennek értékét viszont, mint már rámutattunk, a távolságprofilból kiolvashatjuk:

$$P_b < \left.\frac{\partial T(M,D)}{\partial M}\right|_{M=1, D=\sqrt{4p(1-p)}}$$

Például a már korábban bemutatott Hamming kód teljesítőképességéhez viszonyítva a most megvizsgált konvolúciós kód kemény döntéssel eléri a kódolatlan esetet, de ismert (3,1)-es konvolúciós kód $df = 8$ szabad távolsággal is, ami nyereséget mutat:



Az ábrából az is látszik, hogy a felső korlát kis jel-zaj viszony esetén már nem használható.