DIPLOMAMUNKA

A Belief Propagation algoritmus és alkalmazásai

Kói Tamás

Témavezető: Dr. Csiszár Imre
Professzor emeritus
BME Matematika Intézet
Sztochasztika Tanszék

BME
2009

Tartalomjegyzék

1. fejezet

Bevezető

A Belief Propagation vagy más néven Sum-Product algoritmus grafikus valószínűségi modelleken futó, marginális valószínűségeket meghatározó iteratív eljárás. Az algoritmust számos területen alkalmazzák különböző formában: biológia, mesterséges intelligencia, képtömörítés, digitális kommunikáció. A Belief Propagation elnevezés beszédes. Az algoritmus lényegét úgy szokták szemléltetni, hogy a grafikus modellünkben minden csúcs egy processzornak felel meg. Az iterációs lépések során a processzorok a kapott információkat továbbítják a szomszédoknak. A Dr. Tusnády Gábortól (Rényi Intézet) származó "pletyka algoritmus" elnevezés találóan leírja az algoritmus lényegét.

Mint ahogy a kettős elnevezés is sugallja, az algoritmus hosszú fejlődés eredménye, nem köthető egyetlen személyhez. Az alapötlet annyira természetes, hogy számos helyen megjelent, míg elnyerte mai többé-kevésbé egységes formáját. A fejezetben összefoglalom a történetét. Tekintve, hogy nagyon sok hivatkozással rendelkező algoritmusról van szó, a leírás szubjektív. Elnézést kérek azoktól, akiket nem említettem.

Története első állomásának Gallager 1963-ban megjelent [1] PHD dolgozatát tekinthetjük. Ebben Gallager bevezeti az LDPC (Low density parity-check codes) kódcsaládot kiegészítve iteratív dekódoló eljárással. Maga az iteratív eljárás a dolgozatban még nincs szorosan összekapcsolva grafikus képpel, azonban az alapgondolatot már tartalmazza: a globális számítást lokális számításokra vezeti vissza. Gallager eredményétől független, konvolúciós kódok dekódolására kitalált, 1967-ben [2]-ben ismertetett Viterbi algoritmus, és a [3]-ban 1974-ben bevezetett ott kódolásra használt, de általában rejtett Markov modellen is használható Előre-Hátra[1] algoritmus is a Belief Propagation speciális esetei. Mindkét algoritmus kitalálásuk óta széles körben elterjedt. A mai Belief Propagation felé vezető út következő mérföldköve Tanner 1981-es [5] munkája. Ebben bevezeti a Tanner gráf fogalmát kódok grafikus ábrázolására, továbbá a definiált gráfhoz köthetően általánosítja Gallager kódoló eljárásait. Gallager és Tanner eredményei igényes és szép matematikán alapultak, mindazonáltal a kapott kódolási eljárások az akkori technikai színvonalon nem voltak túl hatékonyak a gyakorlatban. Emiatt az említett eredmények sokáig visszhangtalanok maradtak. A Belief Propagation történetében nagyon fontos mérföldkő Berrou, Glavieux, Thitmajshima [7]-ben bevezetett

[1] angol szakirodalomban Forward-Backward algoritmus

Turbó-kódja (1993). Az empirikus eredmények azt mutatták, hogy a konstrukcióval egyes csatornákon, kitalálása előtt nem is remélt mértékben, közel lehetett jutni a Shannon-kapacitáshoz. A Turbó-kód sikerén felbuzdulva sokan kezdték keresni a Turbó-kód sikerének mély okait, ezzel együtt dekódolója működésének matematikai bizonyítását, illetve sokan kerestek hasonlóan eredményes konstrukciókat. Mackay és Neal 1995-ben [9]-ben újra elővették Gallager és Tanner konstrukcióját. Történetileg fontos, hogy ez volt az első információelméleti dolgozat, amelyben szerepel a Belief Propagation kifejezés. A következő általam fontosnak ítélt publikáció Wiberg 1996-os [10] PHD dolgozata. A dolgozatban Wiberg összefoglalta és általánosította Gallager és Tanner eredményeit. Az egyik használt dekódoló algoritmust Sum-Product algoritmusnak nevezte. A későbbiek szempontjából fontos megjegyezni, hogy mindezt a Turbó-kód fényében tette, saját munkája és a Turbó-kód közötti sok fontos kapcsolatra rámutatott.

A Belief Propagation vagy más néven Sum-Product algoritmus történetének másik nem kevésbé fontos szála Pearl 1988-ban megjelent [6] könyvéhez kötődik. Pearl a könyvben foglalta össze addigi kutatásai eredményeit. Az eddig leírtaktól függetlenül bevezette irányított gráf modellek (Bayes-háló) marginális valószínűségeinek számolására a Belief Propagation iteratív algoritmust. Pearl algoritmusa széles körben elterjedt, a könyvben leírt alapgondolatokat számos helyen megtaláljuk.

A fentiekhez hasonló grafikus modelleken működő iteratív algoritmusoknak rengeteg formájuk és alkalmazásuk lett az évek során. Az eredmények hasonlítottak, mégsem mutattak egységes, rendezett képet. Az első szintézist hozó fontos eredménynek Aji és McEliece [11] és [16] cikkei tekinthetőek (1997, 2000). A publikációkban igyekeztek általános keretet adni a különböző iteratív algoritmusoknak. Fontos McEliece és Mackay [12] közös munkája is, melyben sikerült a Turbó-kód dekódoló algoritmusát Pearl Belief Propagation algoritmusának speciális eseteként leírni. Ezekkel a munkákkal részben párhuzamosan folyt Kschischang, Frey és Loeliger szintézist kereső munkája, melyet 2001-ben jelentettek meg [18]-ban. Ebben a Tanner-Gráfokat általánosítva bevezették a Faktor-Gráfnak nevezett grafikus reprezentációt, illetve az új általános reprezentációval kompatibilisen általánosították a kódokon működő Sum-Product algoritmust. A cikkükben belátták, hogy számos iteratív algoritmus speciális esete az általuk definiált általános algoritmusnak: Pearl Belief Propagation algoritmusa, Turbó-kód dekódoló algoritmusa, LDPC kód egyes dekódolói, Előre-Hátra algoritmus, Viterbi algoritmus, Kalman szűrő, egyes Gyors Fourier Transzformációk. A cikk nyomán általánosan elterjedt a Faktor-Gráf reprezentáció. Azonban az algoritmusra a cikkükben használt Sum-Product elnevezés osztozik a Pearl munkájára utaló Belief Propagation elnevezéssel. Én az általánosításokhoz jobban illő volta miatt, továbbá az alapgondolatot nagyon hű kifejezése miatt adtam a dolgozatnak a Belief Propagation algoritmus és alkalmazásai címet. Mindazonáltal a dolgozatban használom a Sum-Product elnevezést, sőt a Belief Propagation/Sum-Product elnevezést is.

A Belief Propagation/Sum-Product algoritmus fa Faktor-Gráfokon egzakt. Nem fa reprezentáción általános esetben tisztázatlanok a konvergencia kérdései. A dolgozatok egy része igyekszik a nem fa reprezentációból különböző átalakításokkal fa reprezentációt készíteni. A dolgozatok egy másik része pedig igyekszik megvizsgálni, hogy mi történik, ha nem fa reprezentáción fut az algoritmus. A

kérdéskör fontossága kétségtelen, számos empirikusan jónak bizonyuló alkalmazás reprezentációja nem fa (pl. Turbó-kód dekódolója). Másik fontos kutatási irány az algoritmus más matematikai ágakkal való kapcsolatának feltárása, és a kapcsolaton keresztüli általánosítás.

Említettem, hogy a Turbó-kód nagyon közel jutott a Shannon-kapacitáshoz. A főgondolatok felhasználásával a Turbó-kód kitalálása óta még jobb eredmények születtek. Fontos, hogy a Turbó-kód okozta lelkesedés eredményeként Richardson és Urbanke jelentős közreműködésével, LDPC kóddal kiegészítve Belief Propagation/Sum-Product algoritmuson alapuló dekódolóval is sikerült nagyon megközelíteni a Shannon-kapacitást ([20], [21], [22]). [32]-ben úgy fogalmaznak a szerzők, hogy az említett eredményekkel körülbelül 50 év kitartó munka után sikerült elérni a Shannon-kapacitást.

Összefoglalva a leírtakat, azt mondhatom, hogy a Belief Propagation/Sum-Product algoritmus sok ember kemény munkájával nyerte el mai általános formáját. A természetességét támasztja alá számos megjelenése és alkalmazása. Különösen figyelemreméltó, hogy a Shannon-kapacitás elérésének hátterében is ez az algoritmus áll.

Most, hogy az olvasónak van elképzelése a témáról, szeretnék néhány sort írni a diplomamunkámról. Összességében a diplomamunka egy nagyon sok formában megjelenő, sok hivatkozással rendelkező algoritmuscsaládról ad rendezett, áttekinthető képet. Mindezt olyan formában teszi, hogy a témakör iránt érdeklődő olvasó könnyen elsajátíthatja az alapokat, továbbá a részletes irodalomjegyzéknek köszönhetően iránymutatást kap a témakör mélyebb feldolgozásához. Az egységes kép miatt számos helyen egyedi vonásokat tartalmaz a témakör bemutatása. A dolgozatban teszek több, általam fontosnak tartott, kisebb megjegyzést, kiegészítést. A diplomamunkámban azok a megállapítások, amelyek állítás név alatt találhatóak, vagy létező tételek általánosabb formái, vagy a szakirodalom által sugallott, de pontosan ki nem mondott megállapítások. A dolgozatot igyekeztem úgy megírni, hogy rámutassak az algoritmus ütemezésének fontosságára, továbbá a 0-k szerepére.

A dolgozatban a hosszabb szakaszokat fejezetnek, a rövidebbeket a fejezetek részeinek nevezem. A második fejezetben több grafikus modellről, továbbá azok gyakorlati alkalmazásukról írok. A harmadik fejezetben tárgyalom körmentes reprezentációkon Pearl eredeti Belief Propagation algoritmusát, [18] Sum-Product algoritmusát és ezek ekvivalenciáját. A negyedik fejezetben részletesen írok a Sum-Product algoritmus körön futó változatáról. Az ötödik fejezetben kódolási alkalmazásokról lesz szó. Végül a hatodik fejezetben összefoglalom a leírtakat.

A dolgozatban a megértés elősegítése miatt sok ábra található. A nem saját készítésű ábrák pontos származási helyét minden esetben feltüntettem. Sok angol szakkifejezést magyarra fordítottam, az eredeti angol megfelelőjüket az első előfordulásnál a lábjegyzetben megadtam. A címhez kötődő Belief Propagation algoritmus, Sum-Product algoritmus, Max-Product algoritmus elnevezéseket nem magyarosítottam.

2. fejezet

Grafikus modellek

A fejezetben a Belief Propagation/Sum-Product algoritmushoz szorosan köthető grafikus reprezentációkról írok. Közös a grafikus modellekben, hogy gyakorlati feladatok inspirálták kialakulásukat. Dióhéjban arról van szó, hogy egy sokváltozós valószínűségi modellben szeretnénk különböző következtetésekre jutni legtöbbször marginális valószínűségek kiszámolásával. A modellező az együttes eloszlást ismeri, pontosabban tapasztalatai alapján ismertnek feltételezi. A gondot az okozza, hogy a változók magas száma miatt a megfelelő következtetések levonása racionális időben összetett matematika nélkül nem lehetséges. A grafikus modellek az ismert eloszlás olyan lényeges tulajdonságait vizualizálják, melyek általában segítenek a számolásban. Az ismertetett modellek mögött komoly és terjedelmes elmélet húzódik, ezért csak a dolgozat témájához szorosan kapcsolódó aspektusokra térek ki. Az egyszerű tárgyalás érdekében, ha csak külön nem említem, az egész dolgozatban felteszem, hogy a szóban forgó valószínűségi változók véges sok értéket vesznek fel. A fogalmak részben folytonos változók, sőt folytonos és diszkrét változók együttes szereplése esetén is érvényben maradnak. A fejezetet [6] és [29] alapján írtam felhasználva [18]-t is.

Ebben a fejezetben, és az egész dolgozatban a valószínűségi változókat nagybetűvel jelölöm. A kisbetűkkel részben a valószínűségi változók értékkészletének rögzített elemét jelölöm. Továbbá a környezettől függően változót is jelölhetnek, amely értéke a valószínűségi változó értékkészletének valamelyik eleme. Valószínűségi változók értékkészletén értelmezett függvényeknél sokszor magát a valószínűségi változót szerepeltetem a függvény argumentumában.

2.1. Bayes-háló

Bayes-háló alatt együttes eloszlások irányított gráf reprezentációját értjük. Rögzítsünk egy sorrendet a valószínűségi változóinkon: $X_1, \ldots, X_n$. Igaz a következő jól ismert formula:

$$p(X_1, \ldots, X_n) = p(X_n|X_1, \ldots X_{n-1}) \ldots p(X_2|X_1)p(X_1). \tag{2.1}$$

Vizsgáljuk meg a formula jobb oldaláról egy tipikus $p(X_i|X_{i-1}, \ldots, X_1)$ kifejezést. Elképzelhető, hogy a konkrét eloszlás esetén a feltételből egyes változók elhagyhatók, vagyis $\exists\ l < i-1$ és $\exists$

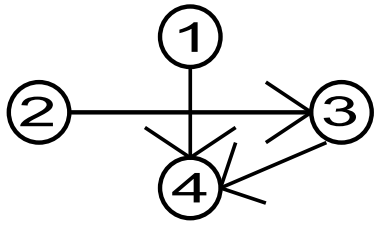
$j_1, \dots j_l$:

$$p(X_i|X_{i-1}, \dots, X_1) = p(X_i|X_{j_1}, \dots, X_{j_l}). \tag{2.2}$$

A feltételben megmaradó valószínűségi változók halmaza nem egyértelmű, például a feltételhez hozzávéve bármelyik kimaradt valószínűségi változót (2.2) igaz marad. Követeljük meg, hogy a tartalmazásra nézve minimális indexhalmazt tartsunk meg. Ez a megkövetelés abban az esetben, ha a p eloszlás szigorúan pozitív[1], egyértelműen meghatározza a megmaradó valószínűségi változók halmazát. Egyértelműen vagy nem egyértelműen, de a (2.1) képlet egyszerűsödik. Példa egy egyszerűsödött formára:

$$p(X_1, X_2, X_3, X_4) = p(X_4|X_3, X_1)p(X_3|X_2)p(X_2)p(X_1). \tag{2.3}$$

Mint látható, az utolsó két tagban az üreshalmazra egyszerűsödött a feltétel. Ezen a ponton érkeztünk el a Bayes-háló fogalmához. A (2.3) egyszerűsödött formához egyértelműen rendelhetünk egy irányított gráfot. Először felrajzolunk egy X_1 változónak megfelelő pontot. Ezt követően felrajzoljuk az X_2-nek megfelelő pontot. Ha az X_2 nem független X_1-től, akkor irányítunk bele egy élet X_1-ből. Általában is ez a szabály, felveszünk a soron következő X_i-nek megfelelő csúcsot, és azon korábbi csúcsokból, amelyek a (2.3) egyszerűsített felbontásban szerepelnek az i-dik változónak megfelelő tag feltételében, irányítunk élet X_i-be. A konstrukciónk eredményeképpen egy irányítottkörmentes irányított gráfot, úgynevezett DAG-ot kapunk. Felhívom a figyelmet, hogy az élek irányításának megszüntetésével lehet kör a gráfban. Továbbá fontos, hogy a konstrukció függ a (2.1) felbontástól,vagyis a csúcsok címkézésének sorrendjétől, továbbá ha a p eloszlás nem szigorúan pozitív, akkor a (2.2) egyszerűsödéstől is. A 2.1 ábrán látható, hogy a példának hozott a (2.3) felbontás milyen Daghoz vezet.



2.1. ábra. A (2.3)-as faktorizációnak megfelelő Bayes-háló

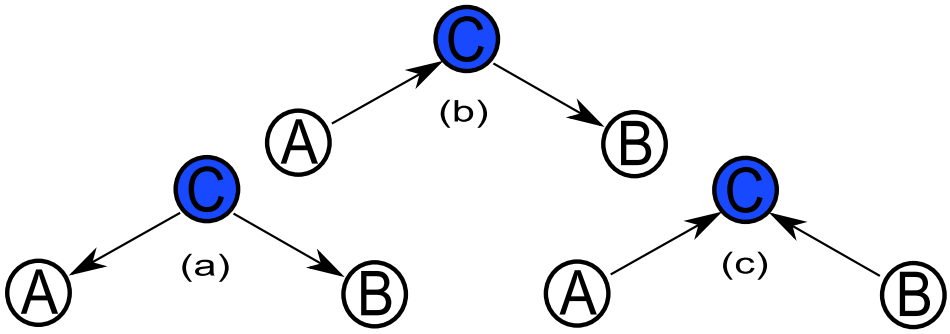
1. Definíció. *Egy G irányított gráfban egy csúcs elődeinek azon csúcsokat nevezzük, amelyek irányított élet küldenek a szóban forgó csúcsba. Egy csúcs utódai azok a csúcsok, amelyekbe élet küld a vizsgált csúcs. Egy csúcs leszármazottai pedig azok a csúcsok, melyekbe irányított úton el lehet jutni a csúcsból.*

2. Definíció. *Ha egy p együttes eloszlásból a fenti konstrukcióval megkapható G irányítottkörmentes irányított gráf, akkor G-t a p eloszláshoz tartozó Bayes-hálónak nevezzük. A Bayes-hálót szokás kiegészíteni a csúcsokhoz rendelt $p(X_i|par(X_i))$ valószínűségeket tartalmazó táblázatokkal, ahol*

[1] minden vektoron 0-nál nagyobb értéket vesz fel

par(X_i) *az* X_i *csúcs elődeinek halmazát jelöli. A táblázatokkal együtt a Bayes-háló egyértelműen meghatározza a modell valószínűségi változóinak együttes eloszlását.*

Vegyük jobban szemügyre a konstrukciót, ezen belül is fókuszáljunk a (2.2) képletre. Gyűjtsük össze az $X_{i-1}, \ldots, X_1$ valószínűségi változókat egy I halmazba, az $X_{j_1}, \ldots, X_{j_l}$ valószínűségi változókat pedig egy J halmazba. Jelöljük K-val az $I \setminus J$ halmazt. Ekkor a (2.2) képlet azt fogalmazza meg, hogy ismerve a J-halmazba tartozó valószínűségi változókat, az X_i valószínűségi változó teljesen független a K-halmazba tartozó valószínűségi változóktól. Vagyis tekinthetünk úgy a Bayes-hálóra, mint feltételes függetlenségi állítások halmazának vizualizációjára. Ezen a ponton érkeztünk el a Bayes-hálók egy újabb fontos tulajdonságához. Pearl [6]-ben bevezet egy d-szeparáció nevű eljárást, aminek segítségével egy konkrét Bayes-hálóról a definiáló feltételes függetlenségi állításokon kívül, könnyű algoritmussal, további feltételes függetlenségi állítások nyerhetők. Nézzük ezt meg részletesebben. Három csúcs egymáshoz viszonyított helyzete háromféle lehet. Megfigyelhetjük a lehetőségeket a 2.2 ábrán. Ha a 2.2 ábra eseteire, mint különálló modellekre gondolunk,



2.2. ábra. Utak mentén a lehetséges találkozási viszonyokat mutatja. A találkozások elnevezése rendre: ki-ki, be-ki, be-be. A színezett csúcsok megfigyelt értékű valószínűségi változókat jelölnek.

akkor elmondható, hogy az (a) és (b) ábrák Markov-láncokhoz tartozó Bayes-hálókat testesítenek meg. Emiatt igaz, hogy C értékének ismerete mellett A és B függetlenek. A harmadik (c) esetben pont az ellenkezője mondható el: A és B függetlenek, de miután C mindkét változótól függ, ezért C ismerete mellett A és B összefüggővé válik. A három eset elnevezése rendre ki-ki[2], be-ki[3], be-be[4]. A Pearl által definiált d-szeparáció tulajdonság így hangzik:

3. Definíció. *A és B változóhalmazokat a C változóhalmaz d-szeparálja, ha minden A-beli csúcsból B-beli csúcsba menő irányítatlan utat blokkol a C halmaz. Blokkolás alatt a következőt értjük:*

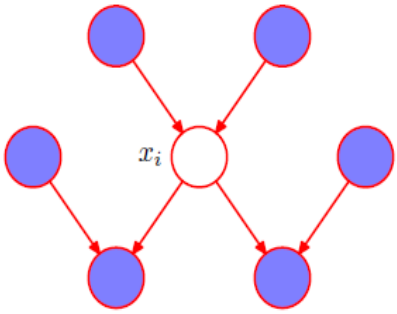
(a) Találunk az út mentén három olyan szomszédos csúcsot, amelyek találkozása ki-ki, vagy be-ki, és a középső csúcs C beli

(b) Találunk az út mentén három olyan szomszédos csúcsot, amelyek találkozása be-be, és sem a középső csúcs, sem a középső csúcs leszármazottai, nincsenek benne C-ben.

[2]angol szakirodalomban tail-to-tail
[3]angol szakirodalomban head-to-tail
[4]angol szakirodalomban head-to-head

1. Tétel. *Egy Bayes-hálóban ha A és B változó halmazokat d-szeparálja C változó halmaz, akkor a C-beli változók ismerete mellett A és B változó halmazok teljesen függetlenek.*

Pearl [6]-ban egy Verma nevű matematikus 1986-os tétele egyszerű következményeként említi a tételt. Bishop [29]-ben azt írja, hogy formális bizonyítást Lauritzen adott 1996-ban. Egy speciális esetet emelek ki. Legyen X_i egy tetszőleges csúcsa a Bayes-hálónak. Rakjuk bele az M-halmazba az X_i csúcs elődeit és utódait, továbbá az utódok X_i-től különböző elődeit. Ekkor az így kapott M-be tartozó valószínűségi változók értékeit ismerve az X_i csúcs teljesen független az összes többi valószínűségi változótól. Az M-halmazt az X_i adott Bayes-hálóhoz tartozó Markov-takarójának[5] nevezzük. Vegyük észre, hogy fontos, hogy beleraktuk a Markov-takaróba az utódok elődeit, hiszen ha nem tettük volna meg, akkor lenne kettő hosszú nem blokkolt út a vizsgált csúcsunkból ezekbe a csúcsokba. A Markov-takarót szemlélteti a 2.3 ábra.



2.3. ábra. [29]-ből származó ábra. Egy kiszemelt X_i változó Markov-takaróját mutatja.

Pearl a Bayes-hálókat elsődlegesen emberek döntési mechanizmusainak modellezésére használta. Ezenkívül jelentős a modell használata a mesterséges intelligencia kutatásban és a biológiában is. Fontosnak tartom a modell orvosi alkalmazásait. Utóbbiaknál a valószínűségi változók sokfélék lehetnek: beteg fizikai paraméterei, szokásai (dohányzás), vizsgálati eredmények, különböző betegségek megléte. Ebben az esetben sok változó értéke ismert (beteg szokásai, vizsgálati adatok), a feladat ezen ismert értékek mellett az egyes betegségekhez tartozó marginális valószínűségek meghatározása. A műveletigény a legegyszerűbb, együttes eloszlást többi változóra összegző algoritmust használva, exponenciális a nem ismert változók számában. Pearl ennek gyorsabb megoldására dolgozta ki a Belief Propagation algoritmust, amely kiaknázza a valószínűségi változó grafikus megjelenését. Az algoritmus pontos ismertetésére a következő fejezetben kerül sor. Annyit megjegyzek, hogy az algoritmus tényleges sebessége az együttes eloszlást definiáló Bayes-háló függvénye.

A rész végén kiemelném az ismertetett grafikus modell azon fontos tulajdonságát, hogy a pontos eloszlás a modellező feltételrendszere. Egyes esetekben megtehető, sőt praktikusabb egy Bayes-hálón keresztül felvenni a modellt. Továbbá vegyük észre, hogy ha a Bayes-háló egy lánc, akkor a valószínűségi változóink egy véges Markov-láncnak felelnek meg. Ilyen nézőpontból tekinthető a Bayes-háló Markov-láncok általánosításának.

[5] angol szakirodalomban Markov-blanket

2.2. Markov-mező

A Markov-mezőt a fizika világában és a képfelismerésben előszeretettel használják. Előbbi nem véletlen, a Markov-mező tekinthető a statisztikus fizikában centrális szerepet játszó Ising modell általánosításának. Dióhéjban együttes eloszlást grafikusan megjelenítő, irányítás nélküli gráf reprezentációról van szó.

4. Definíció. *Tegyük fel, hogy adott $X_1, \ldots, X_n$ diszkrét valószínűségi változók p együttes eloszlása. Legyen adott egy G gráf is, amelynek csúcsai bijektív megfeleltetésben vannak a vizsgált valószínűségi változókkal. A (G, p) párról azt mondjuk, hogy Markov-mező, ha teljesül a lokális Markov-tulajdonság $\forall i$-re:*

$$p(X_i|X_1, \ldots, X_{i-1}, X_{i+1}, \ldots, X_n) = p(X_i|sz(X_i)), \tag{2.4}$$

ahol $sz(X_i)$ a G gráf i. csúcsának a szomszédait jelöli. Vagyis ebben a modellben a Markov-takarót egyszerűen a szomszédok alkotják.

Legyen G egy tetszőleges n csúcsú egyszerű gráf. Tegyük fel, hogy a vizsgált valószínűségi változók együttes eloszlása a következő alakú:

$$p(X_1, \ldots, X_n) = Z^{-1} \prod_{E \in Q} f_E(V_E), \tag{2.5}$$

ahol Z^{-1} normáló konstans, Q a G gráf klikkeinek részhalmaza, V_E az E klikkbe tartozó csúcsoknak megfelelő valószínűségi változók halmaza, f_E pedig ezeken értelmezett nemnegatív függvényt jelöl. Fontos a következő állítás:

1. Állítás. *Tegyük fel, hogy egy p együttes eloszlás előáll a (2.5) alakban egy G gráfhoz köthetően. Ekkor a (G, p) pár Markov-mező.*

Bizonyítás: Jelöljük I-vel a modellben szereplő összes valószínűségi változó halmazát.

$$p(x_i|sz(X_i), I \setminus \{X_i, sz(X_i)\}) = \frac{p(X_1, \ldots, X_{i-1}, x_i, X_{i+1}, \ldots, X_n)}{\sum_{x_i} p(X_1, \ldots, X_{i-1}, x_i, X_{i+1}, \ldots, X_n)} = p(x_i|sz(X_i)). \tag{2.6}$$

Fontos, hogy csak 0-tól különböző valószínűségű feltétellel kell dolgoznunk. Az utolsó lépésben kihasználtuk, hogy mind a számlálóban, mind a nevezőben kiesnek a (2.5) faktorizációban szereplő azon függvények, amelyek nem függnek X_i-től, továbbá miután a függvények klikkeken értelmezettek, az egyszerűsödés után megmaradó függvények csak a szomszédoktól függnek. ■

Szigorúan pozitív függvény esetén a megfordítás is igaz:

2. Tétel. *Hammersley-Clifford tétel: (G, p) pár, ahol p szigorúan pozitív, akkor és csak akkor Markov-mező, ha p előáll a G klikkein értelmezett szigorúan pozitív függvények szorzataként.*

Megjegyzem, hogy ha egy irányítatlan lánc a G gráf, akkor a G gráf olyan együttes eloszlásokhoz tartozhat, melyek a lánc mentén, mint időtengely mentén Markov-láncot alkotnak.

A rész lezárásaképpen egy példát ismertetnék, mely több célt is szolgál: feltárja a kapcsolatot a Markov-mező és az Ising modell között, továbbá a képfelismerés alapötletét is bemutatja.

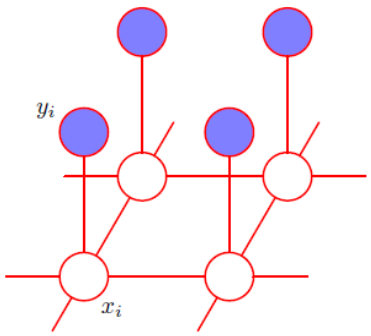
Tegyük fel, hogy rajzoltunk egy fekete-fehér elektronikus képet. A képet úgy kódoljuk, hogy minden pixelnek megfeleltetünk egy X_i valószínűségi változót, melynek értéke 1 ha a pixel fekete, -1 ha fehér. A képet elküldjük ismerősünknek. A küldés közege nem tökéletes, történnek sérülések, ismerősünk y_i pixelcsomagot kap (fixnek tekintjük). A kapott információból szeretné minél pontosabban helyreállítani az eredeti képet. Ennek a megoldására egy általános Markov-mező modellt használ. A (2.5) képlet alapján adjuk meg a Markov-mezőnkent. A klikkek ebben a speciális esetben egyszerűen önálló csúcsok, és élek lesznek. A klikkeken értelmezett pozitív függvényeknek pedig energiát megtestesítő intuitív jelentésük lesz. Ehhez definiáljuk a következő energiafüggvényt:

$$E(X_1,\dots,X_n,Y_1,\dots,Y_n) = \alpha\sum_i X_i - \beta\sum_{(i,j):\text{szom.}} X_iX_j - \gamma\sum_i X_iY_i. \tag{2.7}$$

Az ehhez az energiához tartozó eloszlás a következő:

$$p(X_1,\dots,X_n,Y_1,\dots,Y_n) = Z^{-1}e^{-E(X_1,\dots,X_n,Y_1,\dots,Y_n)}. \tag{2.8}$$

A modellt úgy foglalhatnánk össze, hogy valószínűbbek a kisebb energiájú állapotok. Az energiát pedig három komponens határozza meg. Az első komponens a hozzárendelt paraméter értékétől függően vagy a fehér vagy a fekete pixeleknek ad kevesebb energiát. A második komponens, ha a hozzárendelt paraméter pozitív, akkor az olyan konfigurációknak ad kevesebb energiát, ahol több az olyan szomszédos pixelpár, melyeknek azonos a színük. Az utolsó komponens pedig azt a természetes feltevést tartalmazza, hogy a csatorna nem katasztrofálisan rossz, vagyis kisebb energiájúak az olyan konfigurációk, ahol az eredeti és a kapott pixel színe megegyezik. Ezen általános modell keretein belül a képet fogadó barátunknak nem kell mást tennie, mint a paramétereket racionálisan vagy empirikus adatokra támaszkodva megállapítani, ezt követően pedig megkeresni azt az $X_1,\dots X_n$ konfigurációt, amely maximalizálja a $p(X_1,\dots,X_n|y_1,\dots,y_n)$ valószínűséget. A definiált Markov-mezőhöz tartozó gráf a 2.4 ábrán látható.



2.4. ábra. [29]-ből származó ábra. A (2.5) faktorizációhoz tartozó Markov-mezőt mutatja.

A képtömörítés szakirodalma a feladat megoldására számos közelítő algoritmust tartalmaz. A feladatot mindazonáltal gyorsan és hatékonyan megoldhatjuk, ha a problémát egy részben általánosabb Faktor-Gráfnak nevezett grafikus modell keretein belül vizsgáljuk, és a Faktor-Gráfon a

később definiált Max-Product algoritmus kört is tartalmazó modellre módosított változatát futtatjuk.

2.3. Faktor-Gráf

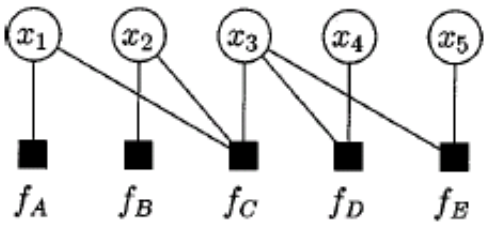
5. Definíció. *Tegyük fel, hogy egy adott egy n változós függvény (értékkészletéről a következő fejezetben lesz részletesen szó), amely több függvény szorzataként előáll:*

$$f(x_1, \ldots, x_n) = \prod_{j \in J} f_j(V_j), \tag{2.9}$$

ahol J egy tetszőleges diszkrét halmaz, V_j pedig a változóknak egy részhalmazát jelöli, f_j pedig tetszőleges V_j argumentumú függvény. Ekkor az f függvény a (2.9) felbontáshoz tartozó Faktor-Gráfja alatt azt a páros gráfot értjük, amelyet a következőképpen konstruálunk: az egyik komponensbe annyi csúcsot teszünk, amennyi változó szerepel a függvény argumentumában, a másik komponensbe pedig minden a (2.9) faktorizációban szereplő f_j függvényhez kapcsolódóan rajzolunk egy csúcsot. Egy változócsúcsot összekötünk egy függvény csúccsal, ha az adott változó szerepel az adott függvény argumentumai között.

A konstrukciót világosabbá teszi a 2.5 ábra. Egy későbbiek szempontjából nagyon fontos technikai definíció:

6. Definíció. *A (2.9) faktorizációhoz kapcsolódóan azt mondjuk, hogy az f függvény előáll a jobb oldalon szereplő függvények szorzataként. Ezentúl függvények szorzatán mindig ezt értem. Speciálisan vektorok is felfoghatók egyváltozós függvényekként, ilyen értelemben vett szorzatuk a koordinátánkénti szorzat.*



2.5. ábra. [18]-ból származó ábra. Az $f_A(x_1)f_B(x_2)f_C(x_1, x_2, x_3)f_D(x_3, x_4)f_E(x_3, x_5)$ faktorizációhoz tartozó Faktor-Gráf.

Hangsúlyozom, hogy a Faktor-Gráf konstrukciójánál nemcsak valószínűségi modellekre korlátozódunk, az f függvény teljesen tetszőleges. Látni fogjuk a következő fejezetben, hogy a [18]-ban leírt általános Sum-Product algoritmus a (2.9)-hez rendelt Faktor-Gráf grafikus tulajdonságait kihasználva az f függvény egyes változóinak marginális $g_{\{x_i\}}$ függvényeit számolja ki:

7. Definíció. *Az f függvény x_i változójának $g_{\{x_i\}}$ marginális függvénye alatt a következőt értjük:*

$$g_{\{x_i\}}(x_i) = \sum_{\sim\{x_i\}} f(x_1, \ldots, x_n), \tag{2.10}$$

ahol $\sim \{x_i\}$ *jelöli az összes* $\{\}$*-ben nem szereplő változókból álló halmazt.*

Miután az előző fejezetben ismertetett Bayes-háló és Markov-mező grafikus reprezentációkhoz társult az együttes eloszlás faktorizációja is, ezért világos, hogy a definiált modellekhez rendelhetünk Faktor-Gráfot. Ez annyit jelent, hogy a különböző modellekben felmerülő feladatokat megoldhatjuk Faktor-Gráfon is.

Nagyon sok cikkben speciális Faktor-Gráffal foglalkoznak, amely nagyon szoros kapcsolatban van a Markov-mezővel. Felteszik, hogy a (2.9) valószínűségi változók együttes eloszlását írja le, továbbá a faktorizációban szereplő minden függvény pontosan kétváltozós. Ebben az esetben legyen G a következő gráf: a csúcsai álljanak bijektív megfeleltetésben a modellben szereplő valószínűségi változókkal, két csúcsot kössünk össze, ha van olyan függvény a (2.9) faktorizációban, ahol közösen szerepelnek. Az így kapott G gráf gyakorlatilag a Faktor-Gráf, annyi a különbség, hogy a függvény csúcsokat eltöröltük. Az 1. Állítás miatt a (G, f) pár Markov-mező. Így ebben a speciális esetben a két reprezentáció ekvivalens.

8. Definíció. *Az előző bekezdésben taglalt speciális esetet Markov-Faktor-Gráf*[6] *reprezentációnak nevezem.*

A negyedik fejezetben lényegesen kiegészítem a Markov-Faktor-Gráfról leírtakat.

A fejezet végén megismétlem azt a bevezetőben már említett tényt, hogy a Faktor-Gráf a kódokra már használt Tanner-Gráf grafikus reprezentáció általánosítása. Tanner-Gráfokról részletesebben írok a kódolási alkalmazásokkal behatóbban foglalkozó fejezetben.

[6] saját elnevezés

3. fejezet

Belief Propagation/Sum-Product algoritmus

Ebben a fejezetben ismertetem Pearl eredeti [6]-ban részletezett, Bayes-hálókon futó, Belief Propagation algoritmusát. Ezt követően a [18]-ban általánosan bevezetett Sum-Product algoritmust ismertetem. Mindkét algoritmust abban az esetben tárgyalom, ha a szóban forgó grafikus reprezentációk nem tartalmaznak irányítatlan köröket. A fejezet utolsó részében megmutatom szintén [18]-ban szereplő rövid vázlat alapján, hogy az ismertetett Sum-Product algoritmus ténylegesen tekinthető az eredeti Pearl algoritmus általánosításának.

Irányítatlankörmentes reprezentációkon az ismertetett algoritmusok egzaktak. Hatékonyságuk azonban attól függ, hogy az együttes eloszláshoz tartozó faktorizáció mennyire bomlott fel részfüggvényekre.

3.1. Pearl Belief Propagation algoritmusa

Tegyük fel, hogy adott egy Bayes-háló, amely nem tartalmaz irányítatlan kört. Mint korábban leírtam az algoritmus során az egyes csúcsokra önálló processzorként gondolhatunk. Tegyük fel, hogy minden processzor tárolja a hozzá tartozó $p(X_i|par(X_i))$ feltételes valószínűségekből álló táblázatot, ahol $par(X_i)$ az X_i csúcs elődeit jelöli. Tegyük fel, hogy néhány valószínűségi változó értékét megfigyeltük. A Belief Propagation algoritmus minden a modellünkben szereplő valószínűségi változóra meghatározza a megfigyelt értékek melletti feltételes marginális valószínűségeket. Mindezt iteratív módon teszi. Minden processzor a számára hozzáférhető információk alapján üzen a szomszéd processzoroknak. Megfelelő számú iterációs lépés után megkapjuk a várt feltételes marginális függvényeket. Speciális esetként, abban az esetben, ha nem rendelkezünk megfigyelésekkel, kapjuk a feltétel nélküli marginális valószínűségeket. Írjuk le a feladatot formálisan.

9. Definíció. *Evidencia halmaznak nevezzük a valószínűségi változóinkra vonatkozó megfigyeléseinkből álló eseményt. A dolgozatban e-vel jelölöm.*

Példával illusztrálom az evidencia halmazt. Tegyük fel, hogy az összes tudásunk az Y és Z valószínűségi változók megfigyelt értékei. Jelöljük ezeket rendre y, z-vel. Ekkor e a következő eseményt jelenti $e = \{Y = y, Z = z\}$.

10. Definíció. *A megfigyeléseink melletti feltételes marginális valószínűségek alatt a* $Bel(x) = p(X = x|e)$ *vektort értjük.*

Néhány technikai elnevezés:

11. Definíció. *Tetszőleges irányított vagy irányítatlan gráfban a csúcs foka a csúcshoz kapcsolódó élek száma (irányítás nem érdekes). Egy irányítatlan gráfban egy csúcs kifoka, a belőle kimenő élek száma, befoka pedig a felé haladó élek száma. Egy csúcsot gyökérnek nevezünk, ha a befoka* 0*, levélnek, ha a kifoka* 0*.*

A feladat megoldásának első lépéseként alkossunk egységes képet. Az olyan változók mellé, amelyek értékeit ismerjük, vezessünk be álválozókat. Tegyük fel, hogy Y-ról tudjuk, hogy az y értéket veszi fel. Ekkor vegyünk fel egy A_Y-al jelölt álváltozót, amely mindig ugyanazt az értéket veszi fel, mint Y. Ennek megfelelően módosítsuk a grafikus reprezentációnkat, vegyünk fel egy A_Y-al jelölt utód nélküli csúcsot, amelynek egyetlen előde az Y. Ezzel együtt az együttes eloszlás is módosul, az Y értékkészletének önmagával vett direkt szorzatán értelmezett függvénnyel, amely 1 ha a két koordináta megegyezik, 0 ha különbözik a két koordináta. Végül tekintsük úgy, hogy igazából A_Y értéke ismert. Az algoritmusban az álváltozók csak küldik az üzenetet, kapni nem kapnak. Tekintve, hogy az álváltozók bevezetése lokális, nem okoz különösebb számításelméleti bonyodalmat az eljárás. Viszont elértük, hogy a kép egységes lett, az evidencia halmazok mindenképpen levelekre vonatkozó információkból állnak. Így minden eredeti változót egységesen kezelhetünk.

Vizsgáljunk egy tetszőleges X változócsúcsot, amely nem álváltozó. Az elődei legyenek $U_1 \dots U_n$, az utódai pedig $Y_1, \dots, Y_m$. Osszuk fel az e evidencia halmazt két részre: e_X^+, e_X^-. Legyen az X utódainak erdője a következő: rakjuk bele X minden utódát, és minden olyan csúcsot, mely az utódaiból elérhető olyan irányítatlan úton, amely nem érinti X-et. A definiált erdőben szereplő álváltozók által meghatározott evidencia halmazt jelöljük e_X^--al. A Bayes-hálónk maradék, az előző erdőn kívüli csúcsokra vonatkozó evidencia halmazát jelöljük e_X^+-al. Továbbá bevezetek egy technikai jelölést, ha egy vektor (vagy komponensei) előtt szerepel a α, az a vektornak azt a normált változatát jelöli, ahol a komponensek összege 1. A levezetésben fontos lesz az irányítatlan körmentességi feltétel. Számos helyen, mikor a d-szeparációra hivatkozom, azért tehetem meg, mert a vizsgált csúcsok között a körmentesség miatt a látott úton kívül más út nem vezet. Ezekkel a jelölésekkel:

$$Bel(x) = p(X = x|e_X^+, e_X^-) = \alpha p(e_X^+, e_X^-|X = x)p(X = x) = \alpha p(e_X^-|x)p(x|e_X^+) = \alpha\lambda(x)\pi(x), \quad (3.1)$$

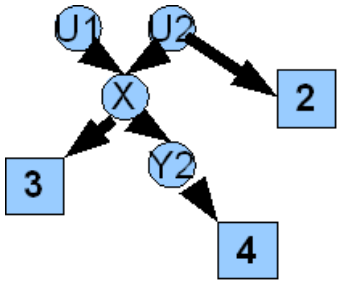
ahol kihasználtuk, hogy az e-hez kötődő valószínűségek konstansnak tekinthetők, továbbá azt, hogy X ismeretében a d-szeparáció miatt e_X^+ és e_X^- függetlenek, és bevezettük a megfelelő tagokra a $\lambda(x)$ és a $\pi(x)$ jelölést. Megjegyzem, hogy a (3.1)-ben, és a továbbiakban is, vektorok koordinátánkénti szorzata szerepel.

Néhány további technikai jelölés következik. Az X-ből kifutó élek mentén további részekre oszthatjuk az evidencia halmazokat:

$$e_X^- = \{e_{XY_1}^- \dots e_{XY_m}^-\}. \tag{3.2}$$

$$e_X^+ = \{e_{U_1X}^+ \dots e_{U_nX}^+\}. \tag{3.3}$$

Ahol $e_{XY_i}^-$ az $X \to Y_i$ él elhagyásával keletkező Y_i-t tartalmazó komponensre (azon csúcsok halmaza, amelyekbe irányítatlan úton el lehet jutni Y_i-ből az eltörölt élet nem használva) megszorított evidencia halmazt jelöli. Hasonlóan $e_{U_iX}^+$ az $U_i \to X$ él elhagyásával keletkező az U_i-t tartalmazó komponensre megszorított evidencia halmazt jelöli. Informatívan az $e_X^{+/-}$ felbontás a rendes változókra használható, míg az élekhez kapcsolódó jelölésben szerepelhetnek álváltozók is. A 3.1 ábrán



3.1. ábra. Illusztráció az evidencia halmazok pontos megértéséhez

körök jelölik az igazi változókat, négyzetek az információt hordozó álváltozókat. Ekkor a következő a helyzet $e_X^+ = \{2\}$, $e_X^- = \{3, 4\}$, $e_{XY_2}^- = \{4\}$, $e_{Y_2}^+ = \{2, 3\}$, $e_{Y_2}^- = \{4\}$, $e_{X3}^- = \{3\}$.

A felvezetett jelölésekkel készen állunk az algoritmus levezetésére. A (3.1) egyenletet fogjuk visszavezetni az elődök és utódoktól kapható üzenetekre.

$$\begin{aligned}
\lambda(x) &= p(e_X^-|x) = \\
&= p(e_{XY_1}^- \dots e_{XY_m}^-|x) = \\
&= p(e_{XY_1}^-|x) \dots p(e_{XY_m}^-|x) = \\
&= \prod_{j=1}^{m} \lambda_{Y_j}(x),
\end{aligned} \tag{3.4}$$

Ahol $\lambda_{Y_j}(x) = p(e_{XY_j}^-|x)$. Az utolsó érdemi lépésben kihasználtuk azt, hogy a d-szeparáció feltétel szerint x ismeretében függetlenek a megfelelő komponensekbe eső valószínűségi változók.

$$\begin{aligned}
\pi(x) &= P(x|e_{U_1X}^+ \dots e_{U_nX}^+) = \\
&= \sum_{u_1,\dots,u_n} p(x|u_1,\dots,u_n, e_{U_1X}^+ \dots e_{U_nX}^+) p(u_1,\dots,u_n|e_{U_1X}^+ \dots e_{U_nX}^+) = \\
&= \sum_{u_1,\dots,u_n} p(x|u_1,\dots,u_n) p(u_1,\dots,u_n|e_{U_1X}^+ \dots e_{U_nX}^+) = \\
&= \sum_{u_1,\dots,u_n} p(x|u_1,\dots,u_n) p(u_1|e_{U_1X}^+) \dots p(u_n|e_{U_nX}^+).
\end{aligned} \tag{3.5}$$

A fenti levezetésben fontos az a tény, hogy az X elődei semmiképpen nem álváltozók, hiszen csak utódok lehetnek azok. Az utolsó előtti és az utolsó lépésben felhasználtuk a d-szeparációt. Az utolsó lépésben speciálisan, az üres halmazra alkalmaztuk a d-szeparációt, azt használtuk ki, hogy az $\{U_i, e^+_{U_iX})\}$-k teljesen függetlenek.

Vezessünk be egy jelölést: $\pi_X(u_i) = p(u_i|e^+_{U_iX})$. Továbbá vastagabb betűvel vektort jelölök. Ezekkel a jelöléssekkel:

$$\pi(x) = \sum_{\mathbf{u}} p(x|\mathbf{u}) \prod_{i=1}^{n} \pi_X(u_i). \tag{3.6}$$

Összegezzük az eddigi eredményeket. A (3.6) és a (3.4) egyenlőségek $Bel(x)$ kiszámolását $\lambda_{Y_j}(x)$-k és $\pi_X(u_i)$-k ismeretére vezetik vissza. Vegyük észre, hogy ha egy Y_i utód egyfokú levél, akkor $\lambda_{Y_i}(x)$ meghatározott. Ebben az esetben ha Y_i álváltozó, akkor igazából a vizsgált X a megfigyelt változó. Legyen a megfigyelt érték x_0, ilyenkor definíció szerint $\lambda_{Y_i}(x)$ az x_0 helyen 1, egyébként 0. Ha Y_i egyfokú levél nem álváltozó, akkor $\lambda_{Y_i}(x)$ az azonosan 1 vektorral egyenlő. Továbbá vegyük észre, hogy ha egy U_i elől egyfokú gyökér, akkor $\pi_X(u_i)$ definíció szerint a $p(u_i)$ vektorral egyenlő. Vagyis ha a $Bel(x)$ kiszámolásához szükséges $\lambda_{Y_j}(x)$-t és $\pi_X(u_i)$-t az X csúcs a szomszédaitól várja, akkor abban az esetben, ha mindegyik szomszéd egyfokú gyökér vagy levél, kész vagyunk. Egyébként pedig ha a szomszédoktól várt információt iteratívan azok szomszédaitól kapott információkra tudjuk visszavezetni, akkor közel járunk a megoldáshoz.

Most átváltunk másik nézőpontba. Továbbra is egy nem álváltozó X csúcsban tevékenykedünk. A jelöléseink a régiek. Azonban most azt vizsgáljuk, hogy a szomszédaink tőlünk várt üzenetét, mely elemekből és hogyan tudjuk felépíteni. Először vizsgáljuk meg az U_i elődünknek küldendő $\lambda_X(u_i)$ üzenetet. Jelöljük V-vel az X U_i-n kívüli elődeinek halmazát. Bontsuk fel az $e^-_{U_iX}$ információs halmazt $\{e^+_{VX}, e^-_X\}$-ra, ahol $e^+_{VX} = \bigcup_{k\neq i} e^+_{U_kX}$. Ekkor:

$$\begin{aligned}
\lambda_X(u_i) &= p(e^-_{U_iX}|u_i) = p(e^+_{VX}, e^-_X|u_i) = \\
&= \sum_x \sum_v p(e^+_{VX}, e^-_X|u_i, v, x) p(v, x|u_i) = \\
&= \sum_x \sum_v p(e^-_X|x) p(e^+_{VX}|v) p(v, x|u_i) = \\
&= \sum_x \sum_v p(e^-_X|x) \frac{p(v|e^+_{VX})}{p(v)} p(x|v, u_i) p(v|u_i) = \\
&= \beta \sum_x \sum_v p(e^-_X|x) p(v|e^+_{VX}) p(x|v, u_i).
\end{aligned} \tag{3.7}$$

A (3.7) levezetésben többször kihasználtunk a d-szeparációból kapható feltételes függetlenséget. Továbbá β-val egy konstanst jelöltünk.

Felhasználva azt a tényt, hogy $p(x|v, u_i) = p(x|\mathbf{u})$, illetve a következő összefüggést:

$$p(v|e^+_{VX}) = \prod_{k\neq i} p(u_k|e^+_{VX}) = \prod_{k\neq i} p(u_k|e^+_{U_kX}) = \prod_{k\neq i} \pi_X(u_k), \tag{3.8}$$

A (3.7) a következő alakra hozható:

$$\lambda_X(u_i) = \beta \sum_x \lambda(x) \sum_{u_k : k\neq i} p(x|\mathbf{u}) \prod_{k\neq i} \pi_X(u_k). \tag{3.9}$$

Vizsgáljuk meg mi a helyzet az utódainak küldött üzenetekkel. Elegáns érvelés következik. $\pi_{Y_j}(x) = p(x|e^+_{XY_j})$-t szeretnénk kifejezni. Tudjuk, hogy $e^+_{XY_j} = e \setminus e^-_{XY_j}$. Ez azt fejezi ki, hogy ha $e^-_{XY_j}$ üres halmaz lenne, akkor $\pi_{Y_j}(x)$ egyenlő lenne $Bel(x)$-el. Rövid időre képzeljük magunkat egy másik Bayes-hálóba, olyanba ahol az $e^-_{XY_j}$ által nyújtott információt nem vesszük figyelembe. Ennek a hálózatnak a $Bel(x)$-ét keressük. Ebben a módosított hálózatban az Y_j-től kapott $\lambda_{Y_j}(x)$ egyenlő az azonosan 1 vektorral. Továbbá ha megnézzük, hogy definíció szerint milyen tartalommal bír az X többi szomszédjától kapott üzenet, akkor látjuk, hogy a módosított hálózatban is pontosan ugyanazt kapja tőlük X. Ezért a (3.1)-t használva:

$$\pi_{Y_j}(x) = \alpha\pi(x)\prod_{k\neq j}\lambda_{Y_k}(x). \tag{3.10}$$

Minden fontos számolnivalót kiszámoltunk. Rakjuk össze a képet. Olvassuk át a következő összefüggéseket: (3.1), (3.4), (3.6), (3.9), (3.10). A (3.1), (3.4), (3.6) a keresett $Bel(x)$ kiszámolását vezeti vissza szomszédoktól várt üzenetekre. A (3.9), (3.10) pedig leírja, hogy a szomszédok a tőlük várt üzenetet vissza tudják vezetni azok további szomszédaitól várt üzenetekre. Mindez lehetővé tesz egy indukciós algoritmust.

Tegyük fel, hogy a csúcsoknak megfelelő processzorok szinkronizáltak, és ha cselekvőképesek, cselekednek. Ezen azt értem, hogy kattog egy közös óra, és minden processzor, minden kattanáskor, minden olyan élen küld információt, amelyhez minden információ a rendelkezésére áll a fent levezett összefüggések alapján. Mint korábban megjegyeztem, minden egyfokú levél és gyökér kezdettől fogva képes információt küldeni. Definiáljuk úgy az algoritmust, hogy az egyfokú csúcsok küldjék mindentől függetlenül, és időben változatlanul azt az információt, amit a szomszédaik várnak tőlük. Két indukcióval belátjuk, hogy a bekezdésben definiált algoritmus megfelelő.

Érthetőség szempontjából fontos megjegyezni, hogy az éleken futó üzeneteknek a normálása nem lényeges az algoritmus során. Bárhogy is normálunk, $Bel(x)$ kiszámolásánál mindig a kívánt eredményre jutunk. Pearl megjegyzi azonban, hogy a normálás fontos lehet, ha az algoritmust számítógépre implementáljuk. Megfelelő normálással csökkenthetőek a numerikus hibák. Most lássuk az ígért bizonyítást.

2. Állítás. *Ha egy csúcs egy adott élen elkezd információt küldeni, azt követően minden lépésben elküldi ugyanazt az információt, továbbá ez az információ épp az, amit a szomszédok várnak tőle.*

Bizonyítás: Az algoritmus kezdetekor az egyfokú csúcsok üzennek a szomszédaiknak, és definíció szerint ez az üzenet időben változatlan, és pont az, amire a szomszédoknak szüksége van. Tegyük fel, hogy az i. időpillanatban üzenni kezd egy csúcs egy adott élen. Ezt azért teheti meg, mert a többi belőle kimenő éleken kapott információt. Ezekből készíti el az üzenetet. Indukciónk miatt a többi élen kapott információ állandó. Továbbá fontos, hogy az adott élen elküldött információt nem befolyásolja az élen visszajövő információ. Így az ezen az élen haladó információ is időben állandó, és épp az, amit az él másik végén levő szomszéd vár. ■

12. Definíció. *Egy irányítatlan vagy irányított gráf $a(G)$ átmérőjének nevezem a gráf leghosszabb irányítatlan útjának élszámát.*

Ha adott egy irányítatlan fa, akkor a következő konstrukcióval áttekinthetőbbé tehetjük. A konstrukciót megelőző kiinduló fát régi fának, a konstrukció során épülő fát új fának nevezem. Választunk egy tetszőleges csúcsot, amelyet az új fa gyökerének[1] nevezünk. Ez az egy csúcs alkotja az új fa 0. szintjét. Ezt követően, ha már egy szintig kész az új fa, akkor a következő szintet úgy konstruáljuk, hogy a megelőző szint csúcsainak megkeressük a szomszédait a régi fában, majd ezek közül azokat, amelyek még nem szerepelnek az új fában, helyezzük az új szintre, majd a régi fában meglévő összeköttetéseket az új fában is behúzzuk. Mindezt addig folytatjuk, amíg van mit az új szintre helyezni.

3. Állítás. *Jelöljük a Bayes-hálónkat F-el. Ekkor legfeljebb $a(F)$ időegység után minden csúcs minden szomszédos élen elkezd üzenetet küldeni. Így az $a(F)$. időegység után minden csúcs ki tudja számolni a saját $Bel(x)$ vektorát.*

Bizonyítás: Ha a Bayes-hálónkból eltüntetjük az irányítást, akkor egy fát kapunk. Válasszuk az irányítás nélküli fában gyökérnek a leghosszabb út egyik felezőpontját. A gyökér megválasztása után a fánk egyértelműen szintekre tagozódik. Szintek szerinti indukcióval belátható a kívánt állítás. A pontos értéket úgy kapjuk meg, ha külön megvizsgáljuk azt az esetet, amikor $a(F)$ páros és azt, amikor páratlan. Mindkét esetben azt kell kihasználni, hogy a gyökértől nem lehetnek távol pontok, mert a leghosszabb út egyik felezőpontjáról van szó. ■

Látjuk, hogy a definiált algoritmus működik. A fenti algoritmus a megértést leginkább segítő változat. Az eddig leírtak alapján világos az is, hogy nem szükséges mindig minden csúcsnak információt küldenie, elég, ha a processzorok egyszer elküldik a helyes információt, a kapott üzeneteket pedig tárolják. Továbbá az üzenetküldésnek nem kell párhuzamosan történnie, elég ha minden lépésben egy üzenetküldésre kész csúcs küld üzenetet tetszőleges olyan sorrendben, melyben mindenkire, minden irányban elég sűrűn sor kerül.

A későbbiek szempontjából fontos, hogy ha rátekintünk az algoritmusra, akkor látjuk, hogy az együttes eloszlásban szereplő 0-k nem befolyásolnak semmit, ha a $p(A|B)$ feltételes valószínűséget 0-nak definiáljuk, ha a feltétel valószínűsége 0. Ezzel csak azt a trivialitást fejezem ki, hogy a teljes valószínűség tételének használatakor nem szummázunk a 0 valószínűségű események szerint. Másik fontos észrevétel, hogy a megfigyeléseink lehetnek olyanok, hogy az egyes változók értékkészletük egy részhalmazába esnek. Ebben az esetben ha az álváltozók által küldött vektorok olyanok, hogy a megfigyelt halmaznak megfelelő koordinátákon 1-t vesznek fel, egyéb helyeken pedig 0-t, akkor az algoritmus tökéletesen működik. A leírt levezetés teljes mértékben érvényben marad, hiszen nem volt szükségünk olyan állításra, amelyben az álváltozók konkrét értéke szerepet játszott volna, csak a hozzá tartozó valószínűségi változók feltételes függetlenségi viszonyait használtuk ki.

Az algoritmus számolási igényéről azt lehet elmondani, hogy a futási idő két dologtól függ, attól, hogy a Bayes-háló modellhez tartozó együttes eloszlás faktorizációjában szereplő tagokban mennyi valószínűségi változó szerepel, továbbá az egyes változók értékkészletének számosságától.

[1] ez a gyökér fogalom eltér az irányított gráfnál ismertetett gyökér fogalomtól

Szerepeljen a modellünkben n darab valószínűségi változó. Ha a valószínűségi változók értékkészletének számossága n-től független konstanssal felülbecsülhető, továbbá a faktorizáció függvényeinek argumentum száma is konstanssal korlátozható, akkor egy iteráció számítási ideje n-ben lineáris. Kevésbé szép faktorizáció esetén az exponenciális számítási idő is elképzelhető. Ezt a megjegyzés az algoritmus későbbiekben leírt minden változatára vonatkozik.

A részt egy szép képpel zárom. Tegyük fel, hogy a Bayes-hálónk egyensúlyban van, vagyis a fent definiált algoritmus valamilyen formában lefutott. Állítsuk be a processzorainkat úgy, hogy ha változást észlelnek, annak megfelelően, ha tudnak cselekedjenek. Ekkor az új információk felbukkanására reagál a rendszer, az új információk tovaterjednek a rendszerben, végül beáll az új egyensúly.

3.2. Sum-Product algoritmus

Tegyük fel, hogy adott a (2.9) képletnek megfelelő n változós függvény a megfelelő faktorizációval. Továbbá tegyük fel, hogy a faktorizációhoz tartozó Faktor-Gráf nem tartalmaz kört. A Sum-Product algoritmus célja az előző fejezetben definiált $g_{\{x_i\}}(x_i)$ marginális függvények kiszámolása. Az algoritmust általános formájában ismertetem. Tegyük fel, hogy a függvények értelmezési tartománya tetszőleges, az egyszerűség végett véges halmaz, értékkészletük pedig kommutatív félgyűrű.

13. Definíció. *R halmazról azt mondjuk, hogy kommutatív félgyűrű ha R elemein értelmezve van két, kétváltozós művelet $(+, \cdot)$ a következő tulajdonsággal:*

1. *$(R, +)$ asszociatív, kommutatív, van 0-val jelölt egységeleme*
2. *$(R, \cdot)$ asszociatív, kommutatív, van 1-el jelölt egységeleme*
3. *Teljesül a disztributivitás: $\forall a, b, c \in R$-re $a(b + c) = ab + ac$*
4. *$\forall a \in R$-re: $0a = 0$*

Informatívan a kommutatív félgyűrű annyiban különbözik a kommutatív gyűrűtől, hogy összeadásra nézve nem követeljük meg az inverz létezését. Az utolsó feltétel gyűrűben triviális, itt viszont ki kell kötni.

Az algoritmust a megszokott módon ismertetem. Levezetek egy rekurziót, ami aztán kihasználva a faszerkezetet, teljes algoritmussá áll össze. A szemléletes tárgyalás érdekében itt is tegyük fel, hogy minden facsúcs egy processzort képvisel. Kezdjük a vizsgálódást egy általános x-el jelölt változócsúcsnál, amit tekintsünk a Faktor-Gráf gyökerének. A változócsúcsnak megfelelő processzor az x változónak megfelelő marginális függvényt szeretné kiszámolni.

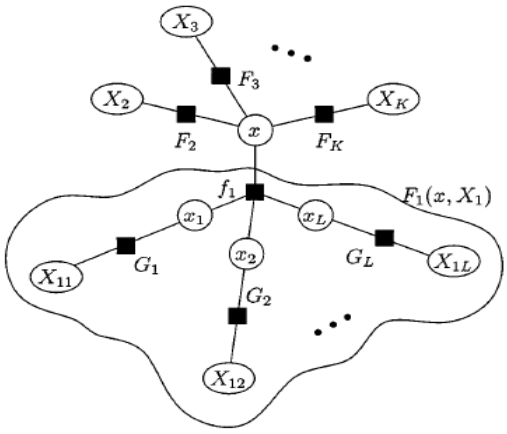
Tegyük fel, hogy x-nek K darab szomszédja van. Felhasználva az asszociativitását, és kommutativitását az eredeti függvényünket csoportosíthatjuk:

$$f(x, x_1, \ldots, x_{n-1}) = \prod_{i=1}^{K} F_i(x, X_i), \tag{3.11}$$

ahol $F_i(x, X_i)$ az x i. szomszédjából induló részfában levő függvények szorzata. Ennek megfelelően X_i pedig a részfában található változók gyűjtőneve. Felhasználva a disztributivitást és azt a tényt, hogy az X_i változóhalmazok diszjunktak, kapjuk:

$$\begin{aligned} g_{\{x\}}(x) = \sum_{\sim\{x\}} f(x, x_1, \ldots, x_{n-1}) = \sum_{X_1}\sum_{X_2}\cdots\sum_{X_K}\left(\prod_{i=1}^{K} F_i(x, X_i)\right) = \\ = \prod_{i=1}^{K}\left(\sum_{X_i} F_i(x, X_i)\right) = \prod_{i=1}^{K}\left(\sum_{\sim\{x\}} F_i(x, X_i)\right). \end{aligned} \tag{3.12}$$

Látjuk, hogy ha az x csúcs megkapja $\left(\sum_{\sim\{x\}} F_i(x, X_i)\right)$-t a függvénycsúcs szomszédaitól, akkor ki tudja számolni a hozzá tartozó marginális függvényt. Kérdés, hogy a szomszédok a tőlük várt üzenetet vissza tudják-e vezetni az ő szomszédaiktól várt üzenetre. Illetve, hogy az egyfokú csúcsok triviálisan tudják-e szolgáltatni a kért információt. Ha ezeket a kérdéseket sikerül tisztázni, akkor mint a Belief Propagation algoritmusnál, itt is kész vagyunk. A további levezetés a 3.2 ábra jelölései mellett fog történni.



3.2. ábra. [18]-ból származó ábra. A dolgozatban a levezetés ez alapján történik.

A 3.2 ábra jelöléseivel vizsgáljuk meg, hogy az x f_1 változó szomszédja hogyan tudja kiszámolni a tőle várt $\sum_{\sim\{x\}} F_1(x, X_1)$ üzenetet. Az ábrán a nagybetűs függvények és nagybetűs változók összevont függvényeket és változókat jelölnek.

$$\begin{aligned} \sum_{\sim\{x\}} F_1(x, X_1) = \sum_{\sim\{x\}} f_1(x, x_1, \ldots x_L) G_1(x_1, X_{11}) \ldots G_L(x_L, X_{1L}) = \\ = \sum_{x_1,\ldots,x_L} f_1(x, x_1, \ldots x_L)\left(\sum_{X_{11}} G_1(x_1, X_{11})\right)\ldots\left(\sum_{X_{1L}} G_L(x_L, X_{1L})\right) = \\ = \sum_{\sim\{x\}}\left[f_1(x, x_1, \ldots x_L)\left(\prod_{i=1}^{L}\sum_{\sim\{x_i\}} G_i(x_i, X_{1i})\right)\right] \end{aligned} \tag{3.13}$$

A levezetés végén eljutottunk remélt eredményünkhöz, $\sum_{\sim\{x\}} F_1(x, X_1)$ kiszámolását visszavezettük f_1 x-től különböző szomszédaitól várt üzenetekre. Vegyük észre, hogy $\sum_{\sim\{x_i\}} G_i(x_i, X_{1i})$ kiszámolása épp azt jelenti, mintha $G_i(x_i, X_{1i})$ függvény Faktor-Gráfjában kíváncsiak lennénk az x_i

változó marginális függvényére. Vagyis a (3.12) és a (3.13) egyenletekkel a feladatunkat ténylegesen kisebb részfeladatokra vezettük vissza, ahol is felírható az említett egyenletek megfelelője.

Ahhoz, hogy algoritmust nyerjünk meg kell nézni, hogy mi történik, ha elértük a fa végét. Ha a két fázisú iterációnk első fázisában egy egyfokú függvényhez érünk, akkor a (3.12)-ben szereplő $\sum_{\sim\{x\}} F_i(x, X_i)$ triviálisan megegyezik magával a szomszéd függvénnyel. Ha pedig a második fázisban f_1-hez kapcsolódik egyfokú változócsúcs, akkor a (3.13)-ban aszerint a csúcs szerint nincs szükség csoportosításra. Ezzel ekvivalens, hogy csoportosítunk szerinte is, és a gyűrű egységelemé-ből álló vektort várjuk tőle.

A Belief Propagation algoritmusnál ismertetett konkrét megvalósítás itt is működik. A processzorok szinkronizáltak, és aki tud üzenni, az üzen. Első lépésben az egyfokú csúcsok elküldik triviális üzenetüket. Ezt követően a processzorok a (3.12) és a (3.13)-ból kiolvasható módon cselekednek, vagyis a változócsúcsok akkor üzennek egy élen, ha minden egyéb élen befutott az információ, és a befutott függvények szorzatát küldik tovább. A függvénycsúcsok többet dolgoznak, ők a befutott függvények szorzatához hozzászorozzák saját magukat, és az így kapott szomszédnyi változós függvénynek elkészítik azt a marginális függvényét, amelyik változónak éppen üzenni akarnak. Indukcióval itt is világosan látszik, hogy az algoritmus egzakt, a Faktor-Gráf átmérőjenek megfelelő lépésszám alatt lefut. Megjegyzem, hogy indukcióval könnyen igazolható módon, a küldött üzenetek mindig egyváltozós függvények, a függvény változója épp az élhez kapcsolódó változó. Miután minden csúcs minden élen küld üzenetet, az üzenetek szorzatából minden változócsúcs el tudja készíteni a saját marginális függvényét.

A levezetést áttanulmányozva észrevehető, hogy miután minden élen a helyes üzenet halad nemcsak az egyedülálló csúcsok készíthetik el a marginális függvényeiket egyszerű koordinátánkénti szorzással.

14. Definíció. *Egy a* (2.9) *faktorizációban szereplő* $\{f_{j_1}, \ldots, f_{j_k}\}$ *függvény halmaz Faktor-Gráf-beli lezártja*[2] *alatt azt a részgráfot értjük, amelyet a függvényhalmaz elemeinek megfelelő csúcsok és azok változó szomszédjai feszítenek ki.*

15. Definíció. *Egy a* (2.9) *faktorizációban szereplő* $\{x_{i_1}, \ldots, x_{i_l}\}$ *változó halmaz együttes marginális függvénye a következő:*

$$g_{\{x_{i_1},\ldots,x_{i_l}\}}(x_{i_1}, \ldots, x_{i_l}) = \sum_{\sim\{x_{i_1},\ldots,x_{i_l}\}} f(x_1, \ldots, x_n). \tag{3.14}$$

4. Állítás. *Tetszőleges függvényhalmaz lezártjára igaz, hogy a benne szereplő változók együttes marginális függvényét megkaphatjuk úgy, hogy a lezárt részgráf felé haladó üzeneteket összeszorozzuk és a kapott függvényhez hozzászorozzuk a részgráfban szereplő függvényeket.*

Bizonyítás: Pontosan úgy kell eljárni, ahogy az egy változóhoz tartozó marginális függvények esetében eljártunk, vagyis *a* (2.9) faktorizációt kell a lezárt részgráfból kimenő élek mentén csoportosítani, majd használni kell a disztributivitást. ■

[2] saját elnevezés

A levezetés végén látjuk, hogy kommutatív félgyűrű struktúra tényleg elegendő az algoritmus tökéletes működéséhez. Megjegyzem, hogy a 0 szerepe az alkalmazások miatt fontos.

Legtöbbször a valós számok a szokásos szorzással és összeadással töltik be a kommutatív félgyűrű szerepét valószínűségi modelleket feltételezve. Ekkor a cél az egyes változókhoz tartozó marginális valószínűségek meghatározása. Az üzenetek tetszőleges normálhatjuk, a végeredményt nem befolyásoljuk, ha azt is normáljuk.

Kódolási alkalmazásokban elterjedt, hogy összeadás helyett maximum vétel szerepel. Könnyen ellenőrizhető, hogy az így kapott $(R^+ \cup \{0\}, \max(\cdot,\cdot), \cdot)$ struktúra a 0-val, mint maximum vételre egységelemmel kommutatív félgyűrűt alkot. Jelöljünk ki egy vizsgált X_i valószínűségi változót. Ekkor a célunk a következő vektor meghatározása:

$$g_{\{x_i\}}(x_i) = \max_{\sim\{x_i\}} f(x_1, \ldots, x_n). \tag{3.15}$$

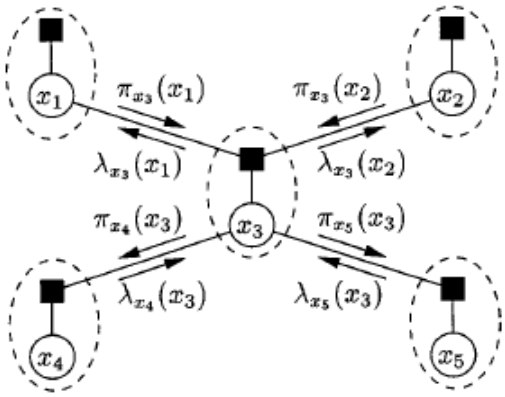
Ekkor a szakirodalom a megfelelő algoritmust Max-Productnak nevezi. Tegyük fel, hogy együttes eloszlásunkban egyetlen konfiguráció[3] a legvalószínűbb. Ekkor a kapott egyváltozós $g_{\{x_i\}}(x_i)$ vektorokat külön maximalizálva változóik szerint, a maximumot beállító értékekből álló konfiguráció épp a legvalószínűbb konfigurációt eredményezi. A 4.3-as részben fontos lesz, hogy ez az állítás visszafelé is igaz. Ha a kapott $g_{\{x_i\}}(x_i)$ vektoroknak egyértelmű a maximuma, akkor csak egyetlen maximális konfiguráció létezik, mégpedig a $g_{\{x_i\}}(x_i)$ vektorok maximumaiból álló konfiguráció. Megjegyzem, hogy Pearl [6]-ban Belief Revision elnevezéssel tárgyalja az algoritmus Bayes-hálón futó változatát.

3.3. Belief Propagation a Sum-Product algoritmus speciális esete

Ebben a fejezetben megmutatom, hogy Pearl eredeti Belief Propagation algoritmusa a [18]-ban szereplő Sum-Product algoritmus speciális esete. Először azt az esetet tárgyalom, amikor a Belief Propagation algoritmust egyszerűen marginális valószínűségek számolására használjuk, vagyis felteszem, hogy nem rendelkezünk megfigyeléssel.

Rendeljünk a (2.2) faktorizációhoz Faktor-Gráfot. A faktorizáció speciális, így a Faktor-Gráfban ugyanannyi a függvénycsúcs, mint a változócsúcs. A Faktor-Gráfban felfedezhető az eredeti Bayes-háló. Állítsuk párba a különböző típusú csúcsokat. Minden változót társítsunk ahhoz a függvényhez, amely az ő feltételes eloszlását írja le az elődei tekintetében. Az ilyen csúcspárok tekinthetőek az eredeti Bayes-háló csúcsainak. Minden a csúcspárok közötti élet irányítsunk a függvénycsúcsok felé. A csúcspárok és az irányított élek együtt épp a (2.2)-höz tartozó Bayes-hálónak felelnek meg. Ezt a megfeleltetést szemlélteti a 3.3-as ábra. Az ábrán Pearl jelöléseivel fel van tüntetve, hogy az éleken milyen üzenetek haladnak a Bayes-hálóban. Ne feledjük, hogy néhány felvetett alternatív lehetőségtől eltekintve mindkét algoritmust úgy definiáltuk, hogy ha egy élen elkezd

[3] az eloszlást leíró p függvény értékkészletének egy eleme



3.3. ábra. [18]-ból származó ábra. A Sum-Product és Belief Propagation kapcsolatát mutatja. A nyilakon fel vannak tüntetve a Belief Propagation üzenetei.

információ áramolni, akkor az helyes információ, és az algoritmus végéig változatlan marad. Így a Sum-Product algoritmus üzenetküldési szabályrendszerét iteratívan használva könnyen látható, hogy a megjelölt üzenetek között épp Pearl képzési szabályainak megfelelő kapcsolat van.

Beláttuk, hogy Pearl eredeti algoritmusának üzenetképzési szabályai levezethetőek a Sum-Product algoritmus üzenetképzési szabályaiból. A teljes leíráshoz hozzátartozik, hogy miután nem rendelkezünk megfigyelt értékekkel, ezért az egyfokú csúcsok ugyanazt az üzenetet továbbítják. Egyetlen különbség a két algoritmus között az a tény, hogy egy Belief Propagation algoritmusban szomszédnak címzett üzenet két Sum-Product lépésben ér célba. A következő fejezetben látni fogjuk, hogy ez a kis anomália bonyolultabb esetekben nem teljesen elhanyagolható.

Most rátérek arra az esetre, ha megfigyelések is rendelkezésünkre állnak. Erről eddig nem írtam a Sum-Product algoritmussal kapcsolatban. Tekintve, hogy most valószínűségi modelleket vizsgálunk, a Sum-Product algoritmusnál erre az esetre a legkézenfekvőbb megoldás, ha behelyettesítjük az együttes eloszlásba a megfigyelt értékeket. Ez nem okoz különösebb számítási bonyodalmat, műveletigénye körülbelül azonos egy iteráció műveletigényével. A behelyettesítés utáni függvény nem eloszlás, de ha az algoritmusban normálunk, akkor ez a tény nem okoz gondot. Ez a módszer csak akkor működik, ha pontosan tudjuk a megfigyelt változó értékét. Fontos, hogy a behelyettesítéssel a Faktor-Gráfból a megfigyelt változó eltűnik, így tekintve, hogy eddig fával dolgoztunk, a gráf elveszti összefüggőségét. Ebben az esetben a különböző komponenseken külön dolgozhatunk.

Másik megközelítésként, Pearl eredeti megközelítésével összhangban, megtehetjük, hogy minden olyan változócsúcshoz felveszünk egy függvényt, amelyről rendelkezünk megfigyeléssel. A felvett függvény egyszerű egyváltozós függvény, amely abban az esetben, ha a tapasztalt értékeket helyettesítjük be 1-et vesz fel, egyébként pedig 0-t. A második megközelítés alapján futtatott Sum-Product algoritmus a kis anomáliától eltekintve egyezik Pearl algoritmusával.

Végezetül szeretném megjegyezni, hogy a Sum-Product algoritmus leírása egyszerűbb és tömörebb Pearl Belief Propagation algoritmusának leírásánál. De épp a tömörség miatt véleményem szerint nem olyan szemléletes, mint Pearl Belief Propagation algoritmusa.

4. fejezet

Sum-Product általában

Ebben a fejezetben az előző fejezetben bevezetett Sum-Product algoritmus azon változatát vizsgálom, amely olyan Faktor-Gráfokon fut, amely nem feltétlenül fa. A szakirodalomban a Loopy Belief Propagation elnevezést is használják az algoritmus általános megnevezésére. Én egy speciális esetet fogok ezzel a névvel illetni.

Elöljáróban azt tudom mondani, hogy megadunk minden élen egy kezdeti üzenetet, és nem törődve azzal, hogy a gráfunk nem fa, az előző fejezetben levezett üzenetküldési szabályoknak megfelelően frissítjük az üzeneteket. Az ott adott levezetés nem érvényes. Látni fogjuk, hogy nem minden esetben lesz konvergens az algoritmus. Sőt, ha konvergenciát tapasztalunk, akkor is csak remélni lehet, hogy a kapott becslések közel vannak a tényleges értékekhez. A gyakorlati alkalmazásokban mindazonáltal sokszor nagyon sikeresnek bizonyult az algoritmus. A kutatók régóta hiszik, hogy fel nem fedezett összefüggések vannak a háttérben. Ugyan rengeteg előrelépés történt, a teljes, mindenre kiterjedő megértés még hiányzik. A fejezet több saját kiegészítéssel az eddig elért alapvető eredményeket mutatja be. A felhasznált forrást a részeknél külön ismertetem, kivéve az első résznél, amely nem köthető konkrét forrásokhoz.

4.1. Loopy Belief Propagation

16. Definíció. *Legyen f $X_1, \ldots, X_n$ véges értékkészletű valószínűségi változók együttes eloszlása, amely a (2.9) mintájára részfüggvények szorzatára bontható. Jelöljük a faktorizációhoz tartozó Faktor-Gráfot G-vel. Párhuzamos ütemezésű Loopy Belief Propagation algoritmusnak nevezzük a G-n futó következő bekezdésben leírt algoritmust.*

A Faktor-Gráf minden egyes éle pontosan egy változócsúcshoz kapcsolódik. A kapcsolódó változó lehetséges értékeinek elemszáma legyen az él dimenziója. Az éleken kezdetben haladó üzeneteket megtestesítendő, rendeljünk minden élhez két tetszőleges, az él dimenziójának megfelelő, nemnegatív valós értékű vektort, melyek komponenseinek összege 1. *Ha az él az X_i változót köti össze az f_j függvénnyel, akkor jelöljük az élhez rendelt vektorokat az információ áramlásával összhangban $\mu^0_{X_i \to f_j}(x_i)$ és $\mu^0_{f_j \to X_i}(x_i)$-vel. Az algoritmusban az éleken küldött üzenetek a következőképpen*

frissüljenek:

$$\mu^{n+1}_{X_i \to f_j}(x_i) = \alpha \prod_{h \in sz(X_i) \setminus \{f_j\}} \mu^n_{h \to X_i}(x_i) \tag{4.1}$$

$$\mu^{n+1}_{f_j \to X_i}(x_i) = \alpha \sum_{\sim\{x_i\}} f_j(V_j) \prod_{y \in sz(f_j) \setminus \{X_i\}} \mu^n_{y \to f_j}(y) \tag{4.2}$$

A fenti képletekben $sz(\cdot)$ az argumentumban szereplő csúcs G-beli szomszédainak a halmazát jelöli. Definíció szerint az üres produktum 1*-el egyenlő. Az algoritmus n iteráció után a következő közelítést biztosítja az egyes marginális függvényekre:*

$$b^n_{\{x_i\}}(x_i) = \alpha \prod_{h \in sz(X_i)} \mu^n_{h \to X_i}(x_i). \tag{4.3}$$

A képletekben α a korábban megszokott jelölés, vektorokhoz az 1 *összegűre normált változatukat rendeli.*

A definícióban együttes eloszlásról, és így nemnegatív valós értékű függvényről volt szó. A műveletek alatt a megszokott valós műveleteket értettem. Felhívom a figyelmet arra, hogy az üzenetek párhuzamosan frissülnek a párhuzamos ütemezés elnevezéssel összhangban. Más algoritmushoz jutunk ha az üzenetek nem egyszerre, hanem meghatározott sorrendben frissülnek.

17. Definíció. *Reguláris ütemezésnek*[1] *nevezzük a következő frissítési eljárást. Az irányított élek*[2] *halmazát csoportokra osztjuk, a közös csoportban levő irányított élek mentén egyszerre történik frissülés. Ezt követően megadunk a csoportokon egy sorrendet, eszerint frissülnek a csoportba tartozó irányított élekhez tartozó üzenetek. Ha egy üzenet frissül, akkor mindig az üzenetet küldő processzorba legfrissebb beérkezett üzenetek alapján teszi ezt meg. Minden iterációban ugyanez ismétlődik. Fontos, hogy egy ilyen iterációnak a számolási igénye ugyanannyi, mint egy párhuzamos ütemezéssel definiált iteráció számolási ideje.*

18. Definíció. *Irreguláris*[3] *ütemezésnek nevezzük azt az ütemezést, amikor nincsenek iterációs lépések. Egyszerűen egyszerre frissülő irányított él csoportok végtelen sorozata adott. A frissülések a sorozat mentén történnek. Azt feltesszük, hogy minden v irányított élhez létezik egy M_v szám, hogy a sorozat bármely egymást követő M_v elemére teljesül, hogy valamelyik ott szereplő csoportban szerepel a v.*

A későbbi egyértelmű hivatkozást megkönnyítendő, a fenti definícióban szereplő párhuzamos ütemezésű algoritmust hívom Loopy Belief Propagation algoritmusnak. Az eltérő ütemezést mindig jelzem. Mindegyik ütemezés mellett akkor mondjuk, hogy konvergens az algortimus, ha minden élen mindkét irányban konvergálnak az üzenetek.

[1] saját elnevezés

[2] bijektív viszonyban vannak az üzenetekkel, vagyis minden élhez két irányított él tartozik

[3] saját elnevezés

A 16. Definíció mintájára az olvasó tetszőleges kommutatív félgyűrűre definiálhatja a köröket is tartalmazó reprezentáción futó változatot. A Max-Product ilyen módosítása különösen fontos. Elkerülendő a sok elnevezéssel járó kényelmetlenségeket, a módosított algoritmust is Max-Productnak nevezem.

A definiált algoritmus kiinduló üzenetei tetszőlegesek. Emiatt ránézésre nem világos, hogy ha fán futtatjuk az algoritmust egzakt lesz-e? A válasz igen, ugyanúgy átmérőnyi lépés után megkapjuk a pontos marginális valószínűségeket. Látható, hogy az egyfokú csúcsoktól a fa belseje felé haladva terjed a helyes információ. Megjegyzem, hogy fán futtatva tetszőleges irreguláris ütemezés mellett véges lépés után megkapjuk a pontos marginális valószínűségeket. Továbbá emlékezzünk vissza, hogy fa esetben megjegyeztem, hogy lehet úgy módosítani az algoritmust, hogy minden élen minden irányban csak egyszer történjen információáramlás. Ha ezen módosulat frissülési sorrendjének megfelelően definiálunk egy reguláris ütemezést, akkor egy iterációval, és így a lehető legkevesebb számolással a helyes eredményre jutunk.

Itt szeretnék egy kis kitérőt tenni, mely reményeim szerint fontos jelenségre mutat rá. Pearl [6]-ban szintén javasolja, hogy irányítatlankörrel rendelkező Bayes-hálókon futtassák az algoritmusát, mintha fáról lenne szó. Fontos kérdés, hogy ha Pearl algoritmusát is párhuzamos ütemezéssel definiáljuk, akkor a kapott algoritmus milyen viszonyban van a fejezetben definiált algoritmussal. Az előző fejezetben definiált megfeleltetés és anomália választ ad a kérdésre. Ha a fejezetbeli algoritmust használjuk, nem kapjuk vissza Pearl algoritmusát. Látható, hogy nem mindig a teljes üzenetek adódnak tovább. A hiányos üzenetek pedig a körök jelenléte miatt mindenre hatással lehetnek. Emiatt nem Pearl algoritmusát reprodukáltuk. Ha viszont a fejezetbeli algoritmust a következő reguláris ütemezéssel hajtjuk végre, akkor megkapjuk Pearl algoritmusát. Két csoportba osztjuk az üzeneteket. Az előbb frissülő csoportba az előző fejezetben definiált csúcspárokon belüli irányított éleknek megfelelő üzenetek kerüljenek. Értelemszerűen a többi irányított él a második csoportba kerül. Ezzel az ütemezéssel pont megkapjuk Pearl eredeti algoritmusát párhuzamos ütemezéssel. Ez rámutat arra a tényre, hogy az ütemezés kulcsfontosságú abban az esetben, ha kört tartalmazó Faktor-Gráfra alkalmazzuk a Sum-Product algoritmust. Kicsit pontosabban, ha csak egy kijelölt feladat megoldása a cél, akkor a grafikus modellek közötti tetszőleges átalakítás hasznos lehet. Ha azonban algoritmusok ekvivalenciáját akarjuk igazolni, óvatosnak kell lennünk, mert a legtöbb eddigi eredmény a legkönnyebben vizsgálható párhuzamos ütemezéshez köthető. A következő részben ki fog derülni, hogy a 0-k szerepe miatt is óvatosan kell bánni az átalakításokkal.

Egy fontos jelenségre szeretnék a rész végén rámutatni. A Faktor-Gráf azon irányított éleit, amelyek változócsúcsból mennek függvénycsúcsba, rakjuk bele egy V halmazba, a W-be pedig a többi irányított élet. Vegyünk fel v_0 és w_0 vektorokat, úgy, hogy a vektorok koordinátáinak a száma $|V| = |W|$ legyen. Minden koordinátára helyezzük el a a részben definiált algoritmusban a koordinátának megfelelő kezdeti üzenetet. Így v_0 és w_0 olyan vektorok, amelyek komponensei szintén vektorok. Az egyes iterációs lépések során keletkező üzeneteket hasonló módon a v_i és w_i vektorokban tároljuk. Ekkor a (4.1) és a (4.2) képletek f és g függvényeket határoznak meg, amelyekre $w_{i+1} = f(v_i)$ és $v_{i+1} = g(w_i)$. Az algoritmus során ezeket iteráljuk. Ha jobban meg-

gondoljuk, akkor voltaképpen két üzenetsorozatunk van, $v_0 \to w_1 \to v_2 \dots$ és $w_o \to v_1 \to w_2 \dots$. Másképp megfogalmazva két f és g váltott használatából adódó iterációnk van, úgy, hogy az egyik v_0-ból, a másik pedig $g(w_0) = v_1$-ből indul. Definiálok egy reguláris ütemezést. Tegyük fel, hogy előbb a V-beli irányított élek mentén történik frissülés, utána a W-beli élek mentén. Ha eszerint az ütemezés szerint futtatjuk az algoritmusunkat, akkor az eredmény az, mintha csak a $g(w_0)$-ból induló iterációt végeznénk el, de azt kétszer akkora sebességgel. Ebből két következtetést vonhatunk le. Ha konvergenciával kapcsolatos eredmények, mint ahogy az eddigi általam olvasott minden konvergenciával foglalkozó cikkben, a kezdeti üzenetektől függetlenek, akkor a két ütemezés ekvivalensnek tekinthető. A másik fontos konklúzió, hogy még ez a legegyszerűbb, az algoritmus lényegével összhangban levő, reguláris ütemezés is eltér a párhuzamos ütemezéstől. A bekezdésben tárgyalt ütemezés annyira fontos, hogy definícióban nyomatékosítom.

19. Definíció. *Az előző bekezdésben definiált reguláris ütemezést speciális ütemezésnek*[4] *nevezzük.*

4.2. Loopy Belief Propagation konvergenciájának kérdésköre pontosan egy kört tartalmazó reprezentációban

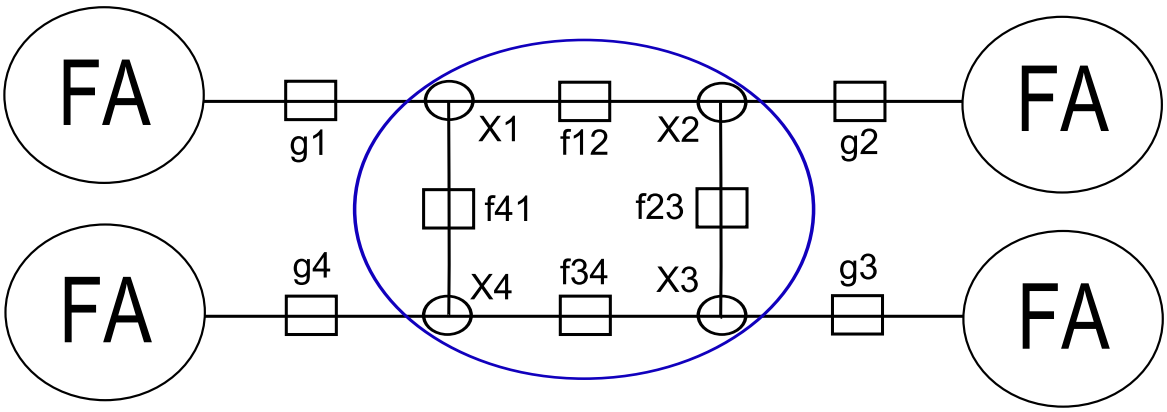
Az alapvető jelenségekre nagyon jól rámutat Weiss 2000-ben megjelent [17] cikke. A dolgozat pontosan egy kört tartalmazó grafikus reprezentációk konvergencia viszonyait vizsgálja. Rengeteg későbbi cikk alapgondolatát megtaláljuk benne. Úgy döntöttem, hogy ezt a cikket alapul véve mutatom be a konvergencia lényegi kérdéseit. A cikk nem Faktor-Gráfon vizsgálja az algoritmust, hanem Markov-Faktor-Gráf reprezentáción. Én általánosabban Faktor-Gráf nézőpontból ismertetem az összefüggéseket. Természetesen szükség lesz különböző feltételekre. A rész kidolgozása közben felhasználtam néhány [13]-ban található gondolatot is. A végeredmény egy precíz, egységes tárgyalás, amely néhány megjegyzést, kiegészítést alapul véve kicsit általánosabb mindkét cikknél.

A pontosan egy kört tartalmazó modellben látni fogjuk, hogy megfelelő feltételekkel előfordulhat, hogy a körök ellenére meglehetősen jó becslést kapunk. Weiss intuitíven mindezt azzal indokolja, hogy az evidenciákat ugyan többször figyelembe vesszük, de minden evidenciát egyenletesen többször veszünk figyelembe.

Tegyük fel, hogy a 4.1 ábrának megfelelő pontosan egy kört tartalmazó Faktor-Gráf adott. Szavakkal elmondva a Faktor-Gráf pontosan egy kört tartalmaz. A körben szereplő n darab változót jelölöm $X_1, \dots, X_n$-el. Ahhoz, hogy az üzenetképzést mátrixszorzással tudjuk reprezentálni fel kell tennünk, hogy a körben szereplő faktorok pontosan két változó függvényei. Ha a körben található függvény az X_i-től és az X_j-től függ, akkor f_{ij}-vel jelölöl. A körben található X_i változóhoz körön kívülről kapcsolódó függvénycsúcsot g_i-vel jelölöm. A 4.1 ábra ezen a ponton csalóka, a változókhoz több körön kívüli függvénycsúcs is kapcsolódhat. Mindez két ponton lényeges, ott kitérek erre.

Technikai megjegyzés, hogy a részben az eddig megszokottól eltérően nem n-el, hanem k-val jelölöm az iteráció sorszámát.

[4] saját elnevezés



4.1. ábra. Pontosan egy kört tartalmazó grafikus reprezentáció. A körben található függvénycsúcsok pontosan két változócsúccsal vannak összekötve.

Első lépésként vegyük észre, hogy néhány lépés után a körön kívüli éleken a kör felé haladó üzenetek változatlanok lesznek. Vizsgálódjunk onnantól, amikortól mindez bekövetkezik. Továbbá ha a körön belüli éleken haladó üzenetek konvergálnak, akkor a körből a körön kívülre küldött üzenetek is konvergálni fognak. Így az algoritmus konvergenciájának vizsgálatához elegendő a belső éleken haladó üzeneteket megvizsgálni. Még egy lépéssel továbblépve, elegendő a belső körben a függvénycsúcstól a változócsúcsig haladó üzeneteket vizsgálni. Ha azok konvergenciát mutatnak, akkor a változócsúcstól a függvénycsúcsig haladó üzenetek is konvergenciát fognak mutatni.

Egymás után használva a (4.2) és a (4.3)-t adódik:

$$\mu^k_{f_{n1}\to X_1}(x_1) = \alpha \sum_{x_n} \left[\mu^{k-2}_{g_n\to X_n}(x_n) \mu^{k-2}_{f_{(n-1)(n)}\to X_n}(x_n) f_{n1}(x_n, x_1) \right]. \tag{4.4}$$

A szokásos vektor reprezentáció helyett reprezentáljuk a körön kívülről érkező üzeneteket mátrixal, vagyis egy négyzetes egységmátrix főátlójába, az egyesek helyett rakjuk bele az üzenet vektort. Az így kapott mátrixot jelöljük D_i-vel. Hangsúlyozom, hogy a vizsgálódást onnan kezdjük, amikortól a körön kívülről a körbe érkező üzenetek változatlanok. Ezért a D_i nem függ a k-tól. Megjegyzem, hogy ha a körön kívülről több függvénycsúcs kapcsolódik a vizsgált változócsúcshoz, akkor a körön kívüli üzenetek koordinátánkénti szorzatát kell D_i-vel jelölni. Ekkor minden eredmény ugyanúgy igaz marad. Továbbá ha az $f(x_n, x_1)$ értékeket értelemszerűen egy M_{n1}: $|X_1| \times |X_n|$[5] mátrixba rakjuk, akkor a következő egyszerű alakra jutunk:

$$\mu^k_{f_{n1}\to X_1} = \alpha M_{n1} D_n \mu^{k-2}_{f_{(n-1)(n)}\to X_n}, \tag{4.5}$$

ahol vektorok és mátrixok, továbbá azok szorzatai szerepelnek. A fenti képletet iterálva a kör mentén kapjuk:

$$\mu^k_{f_{n1}\to X_1} = \alpha M_{n1} D_n M_{(n-1)(n)} D_{n-1} \dots M_{12} D_1 \mu^{k-2n}_{f_{n1}\to X_1} := \alpha C_n \mu^{k-2n}_{f_{n1}\to X_1}, \tag{4.6}$$

ahol C_n a megfelelő mátrixot jelöli. Vegyük észre, hogy erősen kihasználtuk azt a tényt, hogy a körben levő függvények kétváltozósak. A (4.6) rekurzió problémáink egy részét megoldja. A feldolgozott [17] cikk ezen a ponton nem elég precíz. A teljes általánosság precíz bizonyítása érdekében

[5] valószínűségi változó értékkészletének elemszáma szerepel a képletben

[8]-ban lapozgattam, és az ott található 1.18-as és 3.7-es tételek, továbbá a 205. oldal 11-es feladata alapján a következő tétel mondható ki:

3. Tétel. *Hatványmódszer: Legyen A egy komplex elemű mátrix $\lambda_1, \dots \lambda_n$ abszolútértékben csökkenő sorrendbe rakott sajátértékekkel. Ha $|\lambda_1| > |\lambda_2|$, továbbá x_0 indulóvektor olyan, hogy a legnagyobb sajátértékhez tartozó bal oldali sajátvektorral vett belső szorzata nem 0, akkor az $x^{t+1} = \frac{Ax^t}{||Ax^t||}$ rekurzió konvergens, λ_1-hez tartozó jobboldali sajátvektorhoz tart. Továbbá a konvergencia sebessége $\left|\frac{\lambda_2}{\lambda_1}\right|$ (határértéktől való távolság ennyivel szorzódik az iteráció során). Konvergencia alatt, itt az egydimenziós sajátaltérhez való konvergenciát értem.*

Bizonyítás: Vizsgáljuk a $B := \frac{A}{\rho(A)}$ mátrixot, ahol $\rho(A)$ az A mátrix spektrálsugarát jelöli. B-nek van egy darab 1 abszolútértékű egyszeres sajátértéke (μ), a többi sajátérték 1-nél szigorúan kisebb abszolútértékű. Hozzuk B-t Jordan-alakra: $B = SJS^{-1}$, úgy, hogy J első sorának első eleme μ legyen. Vegyük észre, hogy $x^m = \frac{B^m x^0}{||B^m x^0||}$. Látható, hogy $B^m = SJ^m S^{-1}$. J hatványozása úgy történik, hogy a Jordan-alak főátlóban levő blokkjait hatványozzuk. Egyszerű számolással igazolható ([8] 1.18-as tétel bizonyításában ki van számolva), hogy ha a Jordan-blokkhoz tartozó sajátérték abszolútértéke kisebb, mint 1, akkor ennek a sajátértéknek megfelelő sebességgel 0-hoz tart a Jordan-blokk minden cellája (innen adódik a konvergencia sebességére tett kiegészítés), ahogy egyre magasabb hatványra emeljük. Így J^m első sorának első cellája mindig μ többszöröse, a többi cella pedig 0-hoz tart. Tudjuk, hogy S első oszlopa B legnagyobb sajátértékhez tartozó jobboldali sajátvektor (v), S^{-1} első sora pedig B legnagyobb sajátértékéhez tartozó baloldali sajátvektor (w). Így beszorzással adódik, hogy B^m egyre inkább olyan mátrix lesz, amelynek minden oszlopában v szerepel különböző konstanssal megszorozva. A szorzó konstansok pedig épp w megfelelő koordinátái megszorozva μ 1 abszolútértékű sajátérték valamelyik többszörésevel. Innen leolvasható, hogy $x^m = \frac{B^m x^0}{||B^m x^0||}$ a legnagyobb sajátértékhez tartozó egydimenziós sajátaltérhez tart, ha x^0 és w skaláris szorzata nem 0. ■

Megjegyzem, hogy a hatványmódszer szokásos, speciálisabb alakja, hogy felteszik a vizsgált mátrixról, hogy diagonizálható, vagyis, hogy van sajátvektorokból álló bázisa. Az egyszeres domináns sajátértékre vonatkozó feltétel változatlan. A másik feltételt úgy szokták megfogalmazni, hogy ha a kezdeti vektort felírjuk a sajátvektorok bázisában, akkor a domináns sajátvektor együtthatója különbözzön 0-tól. Vegyük észre, hogy ebből a két feltételből következik a fenti tétel megfelelő két feltétele. Az előző jelöléssel $w^\top C = \lambda_1 w^\top$. Legyen v_i λ_i-hez tartozó jobboldali sajátvektor, ahol $i \neq 1$. Ekkor $\lambda_1 w^\top v_i = w^\top C v_i = \lambda_i w^\top v_i$. Ez csakúgy teljesülhet, ha $w^\top v_i = 0$, vagyis w ortogonális v-n kívül az összes jobboldali sajátvektorra. Így abból, hogy a kezdeti vektor v-hez tartozó együtthatója nem 0, következik, hogy a kezdeti vektor nincs benne a w-re ortogonális vektorok alterében (a változatlan másik feltétel teljesülése esetén).

4. Tétel. *[17] tétele: Ha C_n minden eleme pozitív, akkor $\mu^k_{f_{n1}\to X_1}$ C_n abszolút értékben legnagyobb sajátértékhez tartozó sajátvektorhoz konvergál a kezdeti értékektől függetlenül. A konvergencia sebessége a két abszolút értelemben vett sajátérték hányadosa.*

Bizonyítás: Megmutatjuk, hogy a tétel feltételei mellett teljesülnek az előző tétel feltételei. A Perron-Frobenius tétel garantálja, hogy $|\lambda_1| > |\lambda_2|$, és w minden koordinátája pozitív. Így tekintve, hogy a kezdeti vektor mindenképpen nemnegatív, és nem az azonosan 0 vektor, a kezdeti értékre vonatkozó feltétel is teljesül. Fontos, hogy a (4.6) rekurzió igazából $2n$ darab rekurzió, de az előző mondatból következően mindegyik ugyanoda tart. ■

Megjegyzem, hogy ha a körben található minden függvény, továbbá a körhöz közvetlenül kapcsolódó minden függvény szigorúan pozitív, akkor az előző tétel feltételei teljesülnek, függetlenül attól, hogy az egyéb függvények tartalmaznak-e 0-t.

Fontos, hogy tetszőleges irreguláris ütemezésnél a levezetés érvényben marad, az egyetlen különbség, hogy előfordulhat, hogy a (4.6) iterációs képletben a rekurzió mélysége kisebb (itt is kihasználtuk, hogy a körbe érkező információ konstansnak tekinthető egy idő után). A felbukkanó mátrixok minden ütemezés mellett ugyanazok. Így a határérték is ugyanaz. Továbbá az is látszik, hogy a konvergencia nem lehet lassabb, ha reguláris ütemezést használunk, hiszen ekkor az egyes lépések számítási ideje nem változik, viszont a (4.6) rekurzió mélysége bizonyos éleken kisebb lehet, így a konvergencia is gyorsabb. Ne feledjük, hogy az előző rész végén definiált speciális ütemezés is részben alátámasztja az itt tapasztalt jelenségeket (határérték, sebesség). Mindez felveti azt a kérdést, hogy az eltérő ütemezésnél tapasztalt jelenségek igazak-e általában. [30] és [31]-ben találhatóak szimulációk amelyek azt mutatják, hogy egyes irreguláris ütemezésekkel futtatva az algoritmust, az több esetben és gyorsabban konvergál. A szóban forgó jelenség matematikai igazolása az említett cikkekben elkezdődött, azonban az eddig elért eredmények teljesnek semmiképpen nem nevezhetőek. A kérdés fontosságát alátámasztja az ütemezés fontos szerepe a reprezentációk átalakításánál.

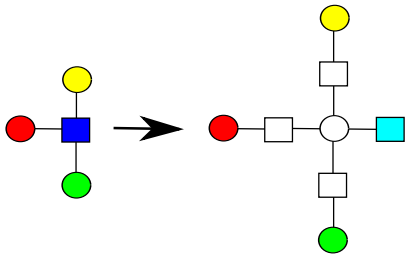
Megjegyzem, hogy akkor is konvergens algoritmushoz juthatunk, ha rendelkezünk megfigyelésekkel. Ha egy körön kívüli csúcsról tudjuk, hogy értékkészletének mely részhalmazába esik, akkor az új faktor felvételével járó eljárással, az algoritmusunk konvergens marad. Továbbá, ha egy körön belüli változónak ismerjük a konkrét értékét, akkor a behelyettesítő eljárással a Faktor-Gráfunk két fa komponensre esik szét, amelyeken teljesen pontosan elvégezhetjük a marginális valószínűségek kiszámolását.

Az eddigi megjegyzéseket összefoglalva igaz a következő állítás:

5. Állítás. *Ha a vizsgált Faktor-Gráf pontosan egy kört tartalmaz, minden a körben szereplő függvény két változó függvénye és minden körben és a körhöz közvetlenül kapcsolódó függvénycsúcs szigorúan pozitív, akkor a Loopy Belief Propagation algoritmus tetszőleges irreguláris ütemezés melletti változata tetszőleges kezdeti üzenet mellett konvergens. A határérték nem függ sem a kezdeti üzenettől, sem az ütemezéstől. Néhány változó ismert értéke mellett is elmondható ugyanez. Reguláris ütemezést használva az üzenetek konvergenciája nem lehet lassabb a párhuzamos ütemezésnél tapasztaltnál.*

Az a tény, hogy a függvénykomponens pontosan két változó függvénye, csak részben jelent megszorítást. Ha a célunk egy konkrét feladat megoldása, akkor átalakíthatjuk a modellünket úgy,

hogy csak kétfokú függvénycsúcsok szerepeljenek benne. Tegyük fel, hogy f_j faktorizációban szereplő függvény $a_1 \dots a_k$ változók függvénye. Ekkor vezessünk be egy új $(\hat{a_1} \dots \hat{a_k})$ összetett változót. Ezzel együtt az f_j függvény helyett vezessünk be új függvényeket. Legyenek $f_{a_i}(a_i, (\hat{a_1} \dots \hat{a_k}))$ függvények olyanok, hogy 1-t vesznek fel, ha $a_i = \hat{a_i}$, egyébként pedig 0-t. Ezenkívül vezessük be f_j kalapos változókon értelmezett replikáltját: $\hat{f_j}((\hat{a_1} \dots \hat{a_k}))$. Könnyű látni, hogy a definiált új faktorizáció ekvivalens a régivel. Továbbá abban a gyakorlatban fontos esetben, amikor a függvények alacsony fokúak, nem okoz komplexitás növekedést a használata, hiszen az átalakítás lokális, és az összetett új változó csúccsal járó számítási nehézségek a régi modellben is jelen voltak az átalakítást kiváltó függvénycsúcsnál. A bekezdésben tárgyalt átalakítás Markov-mezőre vonatkozó változata megtalálható [23]-ban. Az átalakítás és a Markov-Faktor-Gráf kapcsolatáról a következő részben lesz szó. Az átalakítást mutatja be a 4.2 ábra.



4.2. ábra. Illusztráció kettőfokú változócsúcsot létrehozó transzformációhoz. Az azonos színű csúcsok egymásnak felelnek meg. A függvénycsúcsnál a más árnyalat arra utal, hogy replikált változatról van szó.

Ez az átalakítás jelen esetben hasznos. Ha tetszőleges pontosan egy kört tartalmazó grafikus reprezentációban a körben találhatóak olyan függvénycsúcsok, amelyek több változó függvényei, akkor elvégezve az átalakítást, elérhetjük, hogy a körben csak kétváltozós függvények szerepeljenek. Az ok, amiért nem ezzel kezdtem a fejezetet az a tény, hogy ezt az átalakítást használva nem lehet az elégséges feltételeket egyszerűen leírni. A gond az, hogy az f_{a_i} függvények miatt mindenképpen kapcsolódik a körhöz 0-t is tartalmazó függvénycsúcs. A helyzeten enyhít az a tény, hogy ugyan az f_{a_i} függvények tartalmaznak 0-t, de szigorúan pozitív vektorokat szigorúan pozitív vektorokba visznek. Kihasználva azt a tényt, hogy a levezetés nem függött az ütemezéstől (átalakítás a legtöbb esetben ütemezésbeli eltérést von maga után), adódik a következő állítás:

6. Állítás. *Legyen adott egy pontosan egy kört tartalmazó Faktor-Gráf, amelyben a körben található függvények szigorúan pozitívak. Tegyük fel, hogy miután a kör felé haladó üzenetek állandóvá válnak, minden kör felé haladó vektor szigorúan pozitív. Ekkor a Loopy Belief Propagation algoritmus tetszőleges irreguláris ütemezés melletti változata tetszőleges kezdeti üzenet mellett konvergens. A határérték nem függ sem a kezdeti üzenettől, sem az ütemezéstől. Néhány változó ismert értéke mellett is elmondható ugyanez, ha a velük történő munka nem rontja el a feltételt. Reguláris ütemezést használva az üzenetek konvergenciája nem lehet lassabb a párhuzamos ütemezésnél tapasztaltnál.*

A bevezetett jelölések mélyebb megértést tesznek lehetővé. Sajátvektor alatt a továbbiakban mindig jobboldali sajátvektort értek. Tegyük fel, hogy az 5. Állítás feltételei teljesülnek. Vegyük szemügyre a következő levezetést az X_1 változó marginális valószínűségeivel kapcsolatban:

$$\begin{aligned} p(X_1 = i) = \alpha \sum_{x_2,\dots,x_n} & f_{12}(i, x_2) g_2^*(x_2) f_{23}(x_2, x_3) \dots \\ & \dots f_{(n-1)(n)}(x_{n-1}, x_n) g_n^*(x_n) f_{n1}(x_n, i) g_1^*(i) = \\ & = \alpha \sum_{x_2} f_{12}(i, x_2) g_2^*(x_2) \sum_{x_3} f_{23}(x_2, x_3) g_3^*(x_3) \dots \\ & \dots \sum_{x_n} f_{(n-1)(n)}(x_{n-1}, x_n) g_n^*(x_n) f_{n1}(x_n, i) g_1^*(i) = \\ & = \alpha e_i^\top M_{12}^\top D_2 \dots M_{(n-1)(n)}^\top D_n M_{n1}^\top D_1 e_i = \alpha e_i^\top (D_1 C_n D_1^{-1})^\top e_i. \end{aligned} \tag{4.7}$$

A levezetésben e_i az i. egységvektort jelöli. A g_i^* jelentése összetett. Tudjuk, hogy a 4.1 ábrán az x_i-hez a g_i függvényen keresztül kapcsolódik egy fa komponens. A $g_i^*(x_i)$ rögzített x_i érték mellett a fában található függvények x_i-től különböző változóinak szummázásából adódó szám. Más megfogalmazásban a Belief Propagation algoritmus során a fa felől a körbe érkező üzenet. Megjegyzem, hogy akkor sincs probléma, ha az x_i változóhoz több körön kívüli függvénycsúcs kapcsolódik. Ez esetben más a D_i mátrix definíciója, azzal pedig minden összhangban van.

Kihasználva azt a tényt, hogy $(D_1 C_n D_1^{-1})^\top$ főátlója megegyezik C_n főátlójával adódik, hogy $p(X_1 = i) = \alpha C_n(i, i)$.

Az eddig eredményeket felhasználva össze tudjuk kapcsolni az eljárás által szolgáltatott becslést és az igazi marginális valószínűségeket. Jelöljük v-vel C_n abszolút értelemben legnagyobb sajátértékéhez tartozó sajátvektorát, s-sel $(D_1 C_n D_1^{-1})^\top$-ét. Vegyük észre, hogy $(D_1 C_n D_1^{-1})^\top$ épp $\mu^k_{f_{12} \to X_1}$ iterációs mátrixa. Így a határüzenethez tartozó becslés a következő alakot ölti:

$$b_{X_1}(i) = \alpha D_1(i, i) s(i) v(i). \tag{4.8}$$

Írjuk fel $(D_1 C_n D_1^{-1})^\top$ mátrixot $S \Lambda S^{-1}$ Jordan-alakban a főátlóban abszolút értelemben csökkenő sorrendben sajátértékekkel. Ekkor az S mátrix első oszlopa épp s. Felhasználva ezt az alakot:

$$C_n = D_1^{-1} \left((D_1 C_n D_1^{-1})^\top \right)^\top D_1 = (S^{-1} D_1^{-1})^\top \Lambda^\top S^\top D_1. \tag{4.9}$$

Felhasználva, hogy C_n és $(D_1 C_n D_1^{-1})^\top$ sajátértékei megegyeznek, kapjuk, hogy $S^{-1} D_1^{-1}$ első sora megegyezik v-vel. Így a (4.8)-ból adódik:

$$b_{X_1}(i) = \alpha S(i, 1) S^{-1}(1, i). \tag{4.10}$$

Ezt használva:

$$\begin{aligned} p(X_1 = i) &= \alpha e_i^\top (D_1^{-1} C_n D_1)^\top e_i = \\ &= \frac{e_i^\top (D_1^{-1} C_n D_1)^\top e_i}{Trace((D_1^{-1} C_n D_1)^\top)} = \\ &= \frac{e_i^\top S \Lambda S^{-1} e_i}{Trace(C_n)} = \\ &= \frac{\sum_j S(i,j) \lambda_j S^{-1}(j,i)}{\sum_j \lambda_j} = \\ &= \frac{\lambda_1 b_{x_1}(i) + \sum_{j>1} S(i,j) \lambda_j S^{-1}(j,i)}{\sum_j \lambda_j}. \end{aligned} \tag{4.11}$$

A levezetésben $Trace(\cdot)$ az aktuális mátrix nyomát jelöli, míg λ_i-k C_n sajátértékeit jelölik. A fenti levezetés utolsó sora fontos tartalommal bír. Mutatja, hogy a becslésünk pontatlan, mindazonáltal ha C_n mátrix legnagyobb sajátértéke dominálja a többit, akkor becslésünk közel lesz az igazi marginális valószínűséghez. Tudjuk, hogy a konvergencia is annál gyorsabb minél dominánsabb a legnagyobb abszolút értékű sajátérték. Így látjuk, hogy a becslés pontossága és a konvergencia sebessége szoros kapcsolatban vannak. Ez a jelenség az empirikus tapasztalatokban is felfedezhető. Néhány további speciális esettől eltekintve, még nem sikerült elméletileg vizsgálni ezt a jelenséget.

A körön kívüli éleken nem érvényes a fenti levezetés, vagyis nincs tétel, ami garantálja, hogy a marginális valószínűségek közel lehetnek a becsült valószínűségekhez. [17] tartalmaz empirikus eredményeket, amelyek azt mutatják, hogy a körön kívüli változócsúcsokhoz tartozó becslések rosszak. A rész elején említett intuíció alapján, ez nem meglepő, a körhöz kapcsolódó részfákban a kör miatt a részfán kívüli evidenciák sokkal többször vannak figyelembe véve.

Mint korábban említettem, a Belief Propagation legnagyobb sikerének a kódolási alkalmazások számítanak. Mutatok egy tényt, ami alátámasztja a kódoláson elért sikereket. Szorítkozzuk a 4.1 ábra által definiált modellre. A (4.10) képletben, ha jobban meggondoljuk az α felesleges, nélküle is normált vektor szerepel a képletben. Ezt használva írjuk át (4.11)-t abban a speciális esetben, ha a vizsgált valószínűségi változóink binárisak. Legyen $i, j \in \{0, 1\}, i \neq j$, ekkor:

$$p(X_1 = i) = \frac{\lambda_1 b_{X_1}(i) + \lambda_2 (1 - b_{X_1}(i))}{\lambda_1 + \lambda_2}. \tag{4.12}$$

Ebből pedig következik:

$$p(X_1 = i) - p(X_1 = j) = \frac{\lambda_1 - \lambda_2}{\lambda_1 + \lambda_2} (b_{X_1}(i) - b_{X_1}(j)). \tag{4.13}$$

Tekintve, hogy a vizsgált esetben $\lambda_1 > \lambda_2, \lambda_1 > 0, (\lambda_1 + \lambda_2) > 0$, levonhatjuk azt a következtetést, hogy megfelelő feltételekkel az algoritmus mindig helyesen adja meg a körön található változók valószínűbb értéket.

Felhívom az olvasó figyelmét [14]-re. Ebben Murphy és társszerzői Pearl eredeti Belief Propagation algoritmusát vizsgálják empirikusan, több gyakorlatban is fontos, kört tartalmazó reprezentáción. Azt tapasztalták, hogy nagyon jól közelítette az algoritmus az eredeti marginális valószínűségeket, abban az esetben, mikor konvergenciát tapasztaltak. Fontos, hogy a cikkben találhatunk

példát arra, amikor az algoritmus a számítógépes teszt alatt nem mutatott konvergenciát. Ráadásul a példa olyan volt, hogy a faktorizációban szereplő függvény szigorúan pozitív volt. Ez elméleti szempontból érdekes, hiszen a fenti levezetés mutatja, hogy megfelelő feltételek mellett az egy kört tartalmazó reprezentációkon ez a jelenség nem fordulhat elő.

Ha megengedjük, hogy a függvények 0-t is felvegyenek bizonyos konfigurációkon, akkor könnyű egyszerű példát mutatni, ahol nem tapasztalunk konvergenciát. Tegyük fel, hogy csak egy körünk van. Ne rendelkezzünk megfigyeléssel. A változóink és függvényeink legyenek egyformák. Ekkor a (4.6) rekurzió azzal ekvivalens, mintha stacionárius Markov-lánc eloszlását vizsgálnánk az idő haladásával. Tudjuk, hogy 0-k jelenléte mellett könnyen tapasztalhatunk oszcillációt.

Összefoglalva a részben leírtakat, azt tudom mondani, hogy a konvergenciával kapcsolatos legalapvetőbb jelenségeket bemutattam. A feldolgozott irodalmaknál nagyobb hangsúlyt fektettem az ütemezésre, a 0-k szerepére, az átalakító transzformációk általános értékelésére, illetve az alkalmazásokban elengedhetetlen megfigyelt változók kezelésére.

4.3. Max-Product köröket is tartalmazó reprezentáción

A részben [17] és [23] alapján a Max-Product algoritmus általános változatáról ismertetek eredményeket. A rész tökéletes megértése érdekében érdemes visszatérni a (3.15) képlethez, és a körülötte levő megjegyzésekhez. Tekintve, hogy kört is tartalmazó reprezentációra alkalmazzuk az algoritmust, nem kapjuk meg a (3.15) képletben szereplő maximum értelemben marginális függvényeket, csak közelítést kapunk. Az n. iteráció utáni közelítést a 16. Definícióval összhangban $b^n_{\{x_i\}}(x_i)$-vel jelölöm. A $b^n_{\{x_i\}}(x_i)$-k maximumait beállító értékek alkotta vektor a legvalószínűbb konfigurációra n. iteráció után adott becslés, b^n-el jelölöm. Ha az algoritmus konvergens, akkor a határüzenetekből számolt becsléseket $b_{\{x_i\}}(x_i)$-vel és b-vel jelölöm. Ebben a részben azt fogom vizsgálni, hogy a Max-Product konvergenciája esetén a kapott b vektor milyen kapcsolatban van a modell maximális valószínűségű konfigurációjával.

A részben Markov-Faktor-Gráfokkal dolgozom. Mint a második fejezetben leírtam egyszerűen olyan Faktor-Gráf modellről van szó, amelyben minden függvény pontosan két változó függvénye. Futtassuk speciális ütemezéssel az algoritmust. Vagyis előbb a változóból függvénycsúcsba menő irányított élek mentén történjen frissítés, utána pedig a többi irányított élen. Tekintve, hogy minden függvénycsúcs másodfokú, ezzel az ütemezéssel gondolhatunk egy iterációra úgyis, hogy a változók üzennek egymásnak, és a postázandó üzenetet csak részben számolják ki ők, közbülső lépésként a függvénycsúcsok segítenek számolni. Ezzel a kiegészítéssel gondolhatunk a speciális ütemezés melletti Max-Product algoritmusra, mint Markov-mezőn futó párhuzamos ütemezéssel definiált algoritmusra. Ha Markov-Faktor-Gráfon futtatom a Belief Propagation valamelyik változatát, akkor ezen, ha csak nem említem, azt értem, hogy a Markov-mezőn futtatjuk az algoritmust párhuzamos ütemezés mellett.

Kicsit árnyalom a Markov-Faktor-Gráf reprezentációt. Minden a Markov-Faktor-Gráf modellben szereplő változóhoz kapcsoljunk egy, Pearl eredeti hozzáállásával rokon, álváltozót. Természe-

tesen a faktorizáció is bővül, minden (változó, álváltozó) párhoz köthetően bevezetünk egy új kétváltozós függvényt. Így továbbra is Markov-Faktor-Gráf modellünk van. Megfelelően definiálva az álváltozó és a hozzá kapcsolódó igazi változó közötti függvényt, elérhető az egyváltozós függvények kezelése (álváltozó értékétől független függvény) és a megfigyelt értékek interpretálása is (determinisztikus függvény). Az álváltozók itt sem kapnak üzenetet, csak küldenek. Grafikus megjelenítésnél egyszerűen csak az igazi változókat szokták lerajzolni. A részben X_i-vel jelölöm az igazi változókat, Y_i-vel az álváltozókat. A konstrukció eredményeképpen adott $p(X_1, \ldots, X_n, Y_1, \ldots, Y_n)$ együttes eloszlás. Az algoritmusok futásánál az Y_i-k értékét rögzítettnek tekintjük, legyenek $y_1, \ldots, y_n$, ekkor $p(X_1, \ldots, X_n, y_1, \ldots, y_n)$ az $X_1, \ldots, X_n$ valószínűségi változók $y_1, \ldots, y_n$ értékek melletti feltételes eloszlásának konstans szorosa. A Max-Product algoritmust futtatva a modellen célunk az álváltozók megfigyelt értékei mellett az igazi változók legvalószínűbb konfigurációjának közelítése. Emlékeztetem az olvasót 2.4 ábrához kapcsolódó gyakorlati feladatra, ahol pont ez volt a célunk.

Megjegyzem, hogy a 4.2 ábrán látható átalakítással, illetve az előző bekezdésben szereplő kiegészítéssel, minden Faktor-Gráf Markov-Faktor-Gráffá alakítható. Természetesen a 0-k kezelésével, illetve az ütemezéssel lépnek fel problémák.

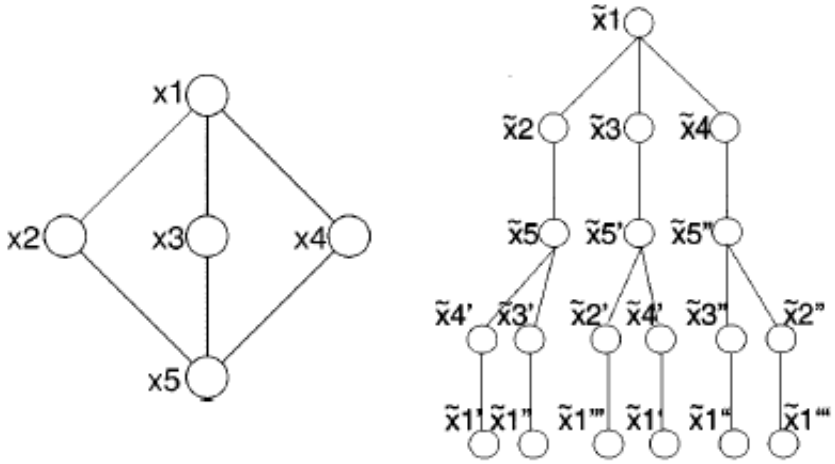
Tehát tegyük fel, hogy a részben definiált árnyalt Markov-Faktor-Gráf modellben dolgozunk. Jelöljük G-vel a hozzá tartozó Markov-mező gráfot az álváltozók nélkül. A továbbiakban egy nagyon hasznos és természetes eszköz kerül bemutatásra, amelynek neve kigombolyított fa[6]. Dióhéjban arról van szó, hogy köröket is tartalmazó reprezentáció helyett egy végtelen fát vizsgálunk, amely lokálisan úgy néz ki, mint a körös reprezentáció. Megjegyzem, hogy ennek az eszköznek a hasznossága nem korlátozódik a Max-Product változatra, később erről még lesz szó. Nézzük meg a pontos definíciót.

20. Definíció. *G Markov-Faktor-Gráfhoz tartozó kigombolyított n szintű fának nevezzük a következő konstrukcióval kapott Markov-Faktor-Gráfot. Válasszunk egy tetszőleges G-beli csúcsot, ennek másolata lesz a kigombolyított fa gyökere. Ezen azt értem, hogy a másolat gyökér és álváltozója közötti függvény legyen pont olyan, mint az eredeti gráfban az eredeti csúcs és álváltozója közötti. Építsük egymás után a fa szintjeit. Az első szinten helyezzük el a gyökérhez tartozó eredeti csúcs minden szomszédjának másolatát, és kössük őket össze a gyökérrel. Az összeköttetésekhez tartozó kétváltozós függvények és az álváltozókhoz köthető függvények legyenek az eredeti gráfban az eredeti változókhoz köthető függvények másolatai. Tegyük fel, hogy az i. szint kész van, ekkor az $(i+1)$. szintet úgy készítjük, hogy az i. szinten levő összes változóra egyenként végrehajtjuk a következő konstrukciót. Megnézzük, hogy a vizsgált változó eredetijének mik a szomszédai az eredeti G gráfban. Ezeknek a szomszédoknak a másolatát egy kivételével a kigombolyított fa következő szintjére helyezzük a szokásos módon. A kivételt a kigombolyított fában vizsgált csúcs elődjének[7] az eredetije jelenti. Őt nem másoljuk le. Ezzel az indukciós lépéssel n. szintig elkészítjük a fát.*

A 4.3 ábra szemlélteti a definícióban szereplő konstrukciót. Technikai könnyítést jelent, ha fel-

[6] angol szakirodalomban unwrapped tree vagy computational tree

[7] a megelőző szinten levő egyetlen csúcs, amivel össze van kötve



4.3. ábra. [23] ábrája. Baloldalon egy Markov-Faktor-Gráf áll, jobb oldalon a hozzá tartozó 4 szintű kigombolyított fa.

tesszük, hogy ha fán futtatjuk a Max-Product algoritmust, akkor ezen az eredeti 3. fejezetben definiált változatát értjük az algoritmusnak (a küldött üzenetek minden esetben helyesek). Az n szintű kigombolyított fa lényege, hogy ha ezen, mint Markov-Faktor-Gráfon futtatjuk a Max-Product algoritmust, akkor a gyökérhez eljutó végső üzenet megegyezik azzal, amit a gyökér G beli eredeti megfelelője kap az eredeti G gráfon futtatott Max-Product algoritmus n. iterációja során. A következő lemma komplikáltnak tűnhet első olvasásra, de egyszerű dologról van szó. Ezt mutatja, hogy a kimondása egyben a bizonyítása is.

1. Lemma. *[17] lemmája*[8]*: Legyen (G, p) Markov-Faktor-Gráf modell tetszőleges. Tegyük fel, hogy a Max-Product algoritmus konvergens a rendszeren. A G-hez tartozó n szintig kigombolyított fát fogjuk vizsgálni, jelöljük F_n-el (kiinduló csúcs legyen tetszőleges). Tudjuk, hogy részfája tetszőleges tovább gombolyított fának. Gombolyítsuk n-él sokkal tovább, egy m szintig a gráfot. Ekkor F_m-en futtatott Max-Product algoritmus F_n-en belül haladó üzenetei nagyon közel lesznek az eredeti modell határüzeneteihez. Megjegyzem, hogy a definícióból nem világos, hogy a lefelé haladó üzenetek is rendben vannak, de tekintve, hogy a konvergens értékek közelében vagyunk ez is teljesül. Így ha módosítjuk F_n legalsó szintjén levő változók és álváltozóik közötti függvényeket, úgy, hogy az álváltozóktól kapott üzenetek imitálják a G_m fában alulról jövő üzeneteket, akkor egy olyan rendszert kapunk, amelyben igaz minden i-re, hogy az x_i másolatok maximum értelemben igazi marginális függvényei közel lesznek a megfelelő $b_{\{x_i\}}(x_i)$ függvényekhez. Az így kapott rendszert jelöljük M_n-el. Dióhéjban összefoglalva olyan fa megjelenésű Markov-Faktor-Gráf modelleket kaptunk, amelyeknek maximum értelemben igazi marginális függvényei közel vannak (m növelésével tetszőlegesen közel) a G-n futtatott konvergens Max-Product algoritmus határbecsléseihez.*

A fenti lemmával közel kerültünk egy szép eredményhez, amely mutatja, hogy a Max-Product általános változatával a maximális konfigurációra kapott becslés lokálisan mindenképpen a legvalószínűbb.

[8]Periodic assignment lemma

21. Definíció. *Legyen (G, p) egy Markov-Faktor-Gráf rendszer. Legyen x vektor egy konfiguráció. Az x kör-fa környezetébe*[9] *azok a z konfigurációk tartoznak, amelyek olyanok, hogy ha S-el jelöljük azon koordináták halmazát, amelyekben különböznek x-től, akkor az S-ben levő koordináták által G-ben kifeszített részgráf olyan, hogy összefüggő komponensei legfeljebb egy kört tartalmaznak. Hangsúlyozom a bijektív megfeleltetést a koordinátahalmazok, és a részgráfok között.*

Megjegyzem, hogy [23]-ban a kör-fa környezet definíciója egy nagyon kicsit speciálisabb, ott a vizsgált részgráf komponensei vagy fák, vagy körök. Azonban a cikkben található Fig.2 ábra (c) része az általam adott enyhén általánosabb definícióra illusztráció. Mindezt úgy foglalhatnám össze, hogy [17] cikket is felhasználva, egy kicsit precízebbé tettem [23]-ban található érvelést.

5. Tétel. *[23] tétele*[10]*: Tegyük fel, hogy adott (G, p) Markov-Faktor-Gráf modell. Továbbá tegyük fel, hogy a Max-Product körös reprezentáción definiált változata konvergens a rendszeren. Tegyük fel azt is, hogy a maximális marginális függvényekre kapott becslések maximuma egyértelmű. Továbbá tegyük fel, hogy az egyértelmű maximumokból álló maximális konfigurációra adott b becslés pozitív valószínűségű a megfigyelések melletti feltételes valószínűségi mezőben. Ekkor teljesül, hogy b kör-fa környezetében levő vektoroknak kisebb a valószínűségük a megfigyelések mellett.*

Bizonyítás: Tegyük fel először, hogy x egy olyan konfiguráció, amely egy a gráfban fának megfelelő koordinátahalmazon való változtatás után megkapható b-ből. Jelöljük az álváltozók alkotta vektort y-al. Az eltérő koordinátákhoz tartozó részfát T-vel. Az x és b konfigurációk egy U részgráfba eső koordinátákra való megszorítását x_U illetve b_U-val jelöljük. Tegyük fel indirekten, hogy y ismerete mellett x valószínűsége nagyobb vagy egyenlő, mint b valószínűsége. Felhasználva azt a tényt, hogy G Markov-mező, utóbbi indirekt feltétel ezt jelenti:

$$p(T = x_T | sz(T) = b_{sz(T)}, y) \geq p(T = b_T | sz(T) = b_{sz(T)}, y), \tag{4.14}$$

ahol $sz(\cdot)$ az argumentumába írt csúcshalmaz szomszédságát jelöli, vagyis azon csúcsok halmazát, amelyekre igaz, hogy szomszédosak valamelyik argumentumbeli csúccsal.

A kigombolyított fa definíciójából világos, hogy ha az n elég nagy, akkor F_n-ben találunk T-vel izomorf $\tilde{T}$[11] részgráfot, úgy, hogy $\tilde{T}$ csúcsai távol vannak az F_n legalsó szintjétől. Ekkor az 1. Lemma miatt, tudunk olyan módosítást végezni F_n-en, hogy a kapott M_n rendszerben az igazi maximális konfiguráció a b vektor másolással kompatibilis változata[12]. Itt kihasználtuk az a tényt, hogy az eredeti gráfon futtatott Max-Product algoritmus marginálisokra adott becsléseinek egyértelmű a maximuma, mert így, a (3.15) képlet alatti megjegyzés miatt, b vektor másolással kompatibilis változata az egyetlen maximális konfiguráció. Ez a tény ellentmond annak, hogy a (4.14)-ből kifolyólag a lokális azonosság miatt a következő teljesül:

$$p(\tilde{T} = \tilde{x}_{\tilde{T}} | sz(\tilde{T}) = \tilde{b}_{sz(\tilde{T})}, \tilde{y}) \geq p(\tilde{T} = \tilde{b}_{\tilde{T}} | sz(\tilde{T}) = \tilde{b}_{sz(\tilde{T})}, \tilde{y}). \tag{4.15}$$

[9] angol szakirodalomban: single loops and trees neighborhood

[10] Claim 1 címke alatt szerepel a cikkben

[11] a hullám a kigombolyított fára utal

[12] minden koordináta a megfelelő M_n csúcs G-beli eredetijéhez tartozó b-beli koordinátával egyezik meg

Most azt az esetet vizsgáljuk, amikor x b-től különböző S koordinátákból álló részgráf egy darab összefüggő komponens, amely pontosan egy kört tartalmaz. Jelöljük a részgráfot is S-el, azon belül a kört pedig L-el. Tekinthetünk úgy a gráfra, mint a körre, amelyhez fák kapcsolódnak. Az indirekt feltevésből most is adódik:

$$p(S = x_S|sz(S) = b_{sz(S)}, y) \geq p(T = b_S|sz(S) = b_{sz(S)}, y), \tag{4.16}$$

Egy kicsit több technikai meggondolást használva, az előzőhöz hasonló módon most is ellentmondásra jutunk. Most is S-hez hasonló alakzatot kell keresnünk az F_n kigombolyított fában. Legyen x_1 az L egyik csúcsa. Tegyük fel, hogy ebből a csúcsból kiindulva gombolyítjuk a gráfot. Ekkor ha elég sokáig gombolyítunk találunk tetszőleges hosszú olyan láncot F_n-ben, amelyben a csúcsok az L csúcsainak a kör struktúrával összhangban levő másolatai. Az ilyen láncon menjünk addig, amíg minden L-beli élen k-szor nem mentünk keresztül. Így mindenképpen x_1 másolatánál állunk meg. A lánc eddig tartó szakaszát jelöljük $\tilde{L}$-al. $\tilde{L}$-t egészítsük ki az L-ből kiinduló fák másolataival. Így egy olyan $\tilde{S}$ részfát kapunk a kigombolyított fában, amely minden S-beli élet pontosan k-szor tartalmaz. A konstrukció miatt igaz a következő azonosság minden k-ra:

$$p(\tilde{S} = \tilde{x}_{\tilde{S}}|sz(\tilde{S}) = \tilde{b}_{sz(\tilde{S})}, \tilde{y}) = \left[p(S = x_S|sz(S) = b_{sz(S)}, y)\right]^k B, \tag{4.17}$$

ahol B egy k-tól nem függő konstans, ami arra utal, hogy a fa két vége miatt néhány tagot külön kell kezelni. Azért nem függ k-tól, mert akármilyen hosszú is az alapláncunk, a fa két vége lokálisan ugyanúgy néz ki. És mivel a (4.17) tetszőleges k-ra igaz, így a (4.16)-t is használva az előző esettel megegyező módon ellentmondást kapunk.

Az utolsó lépés annak igazolása, hogy ha a változtatásban érintett koordináták több előző típusú komponensre bonthatóak, akkor is ellentmondásra jutunk. Ez nem triviálisan, de egyszerűen következik abból, hogy Markov-mezőn vizsgálódunk. ■

Megjegyzem, hogy a bizonyítás akkor is működik, ha 0-k találhatóak az együttes eloszlásban, hiszen a levezetésben a feltételekbe mindig pozitív valószínűségű vektorok kerültek (pozitív valószínűségű konfiguráció rövidítése is pozitív valószínűségű). Továbbá érdemes megfigyelni, hogy több kört is tartalmazó reprezentációkra nem működik a bizonyítás, ugyan lokálisan hasonló alakzatokat ebben az esetben is találhatunk, de a (4.17) képletben szereplő B függne k-tól.

Az ismertetett tétel egyszerű következménye a [17] megfelelő tételének általánosítása arra az esetre, ha az együttes eloszlás tartalmaz 0-t (emiatt volt szükség a kör-fa környezet kicsit eltérő tárgyalására):

7. Állítás. *Legyen adott egy pontosan egy kört tartalmazó Markov-Faktor Gráf. Ekkor ha a Max-Product algoritmus konvergens, és a maximális marginálisokra kapott vektoroknak egyértelmű a maximuma, továbbá maximális konfigurációra kapott becslés pozitív valószínűségű a megfigyelések mellett, akkor végeredményül a megfigyelések melletti legvalószínűbb konfigurációt kaptuk.*

Megjegyzem, hogy [17]-ben szerepel empirikus vizsgálat, amelyben általánosabb modellekben vizsgálódva (nem csak egy kört tartalmazó modellekben) azt az eredményt kapták, hogy a Loopy

Belief Propagation többször konvergál, mint a Max-Product algoritmus, de azokban az esetekben, amikor mindkét algoritmus konvergált, a saját céljukat illetően[13], a Max-Product algoritmus pontosabb becsléseket adott. Utóbbi mondatban beleértem azt a fontos megjegyzést, hogy az empirikus tapasztalatok alapján, a Max-Product konvergencia esetén ugyan legtöbbször jól becsülte a maximális konfigurációt, de nem volt tévedhetetlen.

4.4. Frissebb eredmények

A dolgozat korábbi részeiben leírtam a vizsgált algoritmussal kapcsolatos alapvető fogalmakat, jelenségeket. Ebben a részben újabb eredményeket szeretnék vázlatosan áttekinteni.

[24] 2002-es cikk Markov-Faktor-Gráf modell keretei között, a Loopy Belief Propagation algoritmus konvergenciájára ad elégséges feltételt. A cikkben fel van téve, hogy minden a modellben szereplő függvény szigorúan pozitív. A cikk a Gibbs mértékek elméletét kapcsolja össze a Loopy Belief Propagation algoritmussal. A felhasznált eszköz az előző részben definiált kigombolyított fa végtelen verziója. A fő állítás úgy szól, hogy a végtelen kigombolyított fán a Gibbs mérték egyértelműségéből következik az eredeti gráfon a Loopy Belief Propagation algoritmus konvergenciája. Így Gibbs mérték egyértelműségére létező elégséges feltételekből következik az algoritmus konvergenciája. Megjegyzem, hogy a kapott elégséges feltételek függetlenek a kezdeti üzenetektől, így a 4.1 rész végén található megjegyzés miatt az elégséges feltétel a Faktor-Gráfon futtatott párhuzamos ütemezésű algoritmus konvergenciáját is biztosítja. [30] az előző cikkben elért eredményeket vizsgálja irreguláris ütemezést állítva a középpontba.

[27] dolgozat is Markov-Faktor-Gráf modell keretein belül, szigorúan pozitív függvények esetén vizsgálódik a Loopy Belief Propagation algoritmust illetően. Egy nagyon fontos hiányt pótol. Azt a kérdést vizsgálja, hogy mi történik akkor, ha a küldendő üzeneteket numerikus nehézségek miatt csak közelítőleg lehet továbbítani. A felhasznált eszköztárral végül az algoritmus konvergenciájára is ad elégséges feltételt. A feltétel itt is független a kezdeti üzenetektől.

[28] dolgozat általános Faktor-Gráfon vizsgálódik. Abban az esetben, ha a modellben szereplő függvények szigorúan pozitívak, megmutatja, hogy a (4.1) és a (4.2) képletben definiált iteráció fixpontjai a statisztikus fizikában jól körüljárt szabad energia Bethe közelítő módszerének stacionárius pontjainak felelnek meg. Általánosabban, 0-t is tartalmazó függvények esetén sejtéseket állít fel, intuitív érvelésekkel alátámasztva. Továbbá általánosabb szabad energia közelítő eljárásokon keresztül általánosítja a Loopy Belief Propagation algoritmust. [26] dolgozat [28] cikk korábbi változatait alapul véve, azt vizsgálja, hogy milyen feltételekkel mondható el, hogy a (4.1) és a (4.2) iterációknak pontosan egy fixpontjuk van. Nyitott kérdés, hogy a fixpont egyértelműségéből milyen feltételek mellett következik a Loopy Belief Propagation algoritmus konvergenciája.

A 2007 decemberében megjelent [33] cikk általános Faktor-Gráfon vizsgálja a Loopy Belief Propagation algoritmus konvergenciáját. Nagyon szép, klasszikus matematikát használ. Az eredmények alapvetően szigorúan pozitív függvényekre lettek megfogalmazva, de történt kis előrelépés

[13] más a közelítendő mennyiség az egyes algoritmusoknál

0-k szereplése esetén is. Vizsgálódásainak az alapja a küldött üzenetek kényelmes átparaméterezése, amellyel a (4.1) és a (4.2) rekurziók egyszerűbben kezelhetőek. Az egyszerűbben kezelhető rekurziók által definiált frissítő függvény deriváltját vizsgálva, a Banach-féle fixponttétellel[14] jutott konvergenciát biztosító elégséges feltételekhez. Tekintve, hogy valószínűségi modell keretein belül történik a vizsgálódás, az algoritmus konvergenciájának vizsgálatánál a vektorok normálása érdektelen. Így szükség volt olyan normák bevezetésére, amely érzéketlen vektorok skalárral való szorzására. A cikk az eredmények ismertetésén túl számba vesz régebbi konvergencia eredményeket, és empirikusan összeveti őket a saját eredményeivel. A cikkben elért eredmények empirikusan jobbnak bizonyultak a régebbi eredményeknél. [31] cikk [33] eredményeit továbbgondolva igyekszik vizsgálni a Loopy Belief Propagation algoritmus irreguláris ütemezés melletti változatát. A dolgozatom szellemével nagyon rokon megállapítása a cikknek, hogy az ütemezés kérdésével a cikk megjelenése előtti időszakban fontosságához mérten keveset foglalkoztak.

[36] cikk kéziratát személyesen kapta meg a témavezetőm a cikkek íróitól. Ebben a Belief Propagation algoritmust általános Bregman értelemben információs vetítések sorozataként sikerült leírni. Az elért eredmények a jövőben nagyon hasznosnak bizonyulhatnak.

Az említett cikkek szép és hasznos eredményeket tartalmaznak. Azonban elmondható, hogy az algoritmus konvergencia viszonyainak feltárása közel sem teljes. A 0-k és az ütemezés még több nyitott kérdést szolgáltatnak. Továbbá észrevehető, hogy az eddigi eredmények nagyrészre csak az algoritmus konvergenciájával foglalkozik, arról, hogy a kapott becslés milyen viszonyban van az igazi marginális függvényekkel kevés szó esik.

[14] pontosabban kisebb általánosításaival

5. fejezet

Kódolási alkalmazások

Ebben a fejezetben a Belief Propagation algoritmus kódolási alkalmazásaival foglalkozom. Először ismertetek néhány alapvető fogalmat az információelmélet és az algebrai kódelmélet területéről. Ezt követően ismertetek néhány klasszikus kódolási-dekódolási eljárást a Belief Propagation fényében. Végül a Shannon-kapacitás közelítésében nagyon sikeres kódokról, a Turbó-kódról, és az LDPC kódról lesz szó.

5.1. Kódolási alapfogalmak

Kódolással a matematika két területe, az információelmélet és az algebrai kódelmélet foglalkozik. Nem élesen elhatárolt tudományterületekről van szó, a megkülönböztetés a felhasznált eszközök alapján történik. Az információelmélet a valószínűségszámítás eszköztárát használja nagyon aktívan, míg az algebrai kódelmélet az absztrakt algebra eszköztárát. A dolgozatban az információelmélet oldaláról közelítek. Ezt a részt [4], [15], [25], [34], [35] alapján állítottam össze. Célja a résznek a kódelmélet azon részeinek vázlatos ismertetése, amelyek elengedhetetlenek a Belief Propagation algoritmus kódelméleti jelentőségének megértéséhez.

Arra kérem az olvasót, hogy tanulmányozza az 5.1 ábrát, amelyen Shannon blokkdiagramja található. Alapvetően az a feladat, hogy a forrásból érkező, csatornán keresztülhaladó információt eljuttassuk a felhasználóhoz. Kicsit pontosabban igyekszünk különböző kódoló-dekódoló technikák alkalmazásával a csatornán történő sérülésekkel szemben ellenállóvá tenni az üzenetet. Gyakorlatibb megközelítéssel forrás lehet például egy a Föld körül keringő szonda, amely folyamatosan közvetíti a felhasználó szerepét betöltő GPS helymeghatározó rendszernek az információt. Ekkor a csatorna szerepét a Föld légköre tölti be.

5.1. ábra. Shannon-féle blokkdiagram

Nézzük meg, hogy mindezt hogyan lehet matematika eszköztárával leírni. A forrás matematikai

modellje egy végtelen hosszú $U_1, \ldots, U_k, \ldots$ valószínűségi változó sorozat. Az ismertetni kívánt gráf alakú dekódolók működésének könnyű megértése céljából felteszem, hogy a forrás valószínűségi változói egymástól független, egyenletes eloszlású, bináris értékkészletű valószínűségi változók. Bináris változó alatt a kételemű $\{0,1\}$ testbeli értékű változót értek. A következő speciális csatornamodellt tételezem fel:

22. Definíció. *Azt mondjuk, hogy W egy emlékezet nélküli csatorna bináris bemenettel és $\mathbf{Y} \subset R$ kimeneti ABC-vel, ha adottak $w(y|x), x \in \{0,1\}$ feltételes tömegeloszlások, vagy feltételes sűrűségfüggvények, amelyekkel minden n-re teljesül, hogy $(x_1 \ldots x_n) \in \{0,1\}^n$ bemenet esetén annak a valószínűsége, vagy sűrűségfüggvénye, hogy $(y_1 \ldots y_n) \in \mathbf{Y}^n$ kimenetet kapunk egyenlő:*

$$p(y_1 \ldots y_n | x_1 \ldots x_n) = \prod_{i=1}^{n} w(y_i|x_i). \tag{5.1}$$

Két csatornát szeretnék példaként megemlíteni. A BSC csatornának a kimenete is bináris, és annak a valószínűsége, hogy egy bemenet megváltozik p. Emlékezet nélküli diszkrét Gauss-csatornának hívjuk azt a csatornát, amely esetén $w(y|x)$ σ szórású normális eloszlás, amelynek a várható értéke -1 ha $x = 0$, 1 ha $x = 1$. A dolgozatban említett minden hozzáállás a következő kódoló-dekódoló modellhez vezet:

23. Definíció. *(n,k) paraméterű blokk kódolásnak nevezzük az olyan kódolást, amelyet egy h: $\{0,1\}^k \to \{0,1\}^n$ függvény definiál. A kódolási eljárás a következő, a kódoló megvárja a forrásnak k betűjét, majd a kapott k hosszú blokkot a h függvény segítségével kódolja, majd a már kódolt k darab betűt elfelejti. Ezt ismétli a végtelenségig. Az ilyen kódok rátájának a $\frac{k}{n}$ számot nevezzük, amely azt fejezi ki, hogy egy csatornán áthaladó karakter átlagosan mennyi forrás karaktert tud átvinni. Egy blokk kódhoz tartozó dekódoló egy k: $\mathbf{Y}^n \to \{0,1\}^k$ függvény. A dekódolás eredményeként egy $V_1, \ldots, V_k$ sorozatot kapunk. A dekódolás hibavalószínűségének az $\{U_1, \ldots, U_k \neq V_1, \ldots, V_k\}$ esemény valószínűségét tekintjük. Egy blokk kódolásról azt mondjuk, hogy szisztematikus, ha találunk $i_1, \ldots, i_k$ koordinátákat, hogy minden $x \in \{0,1\}^k$-ra a $h(x)$ megszorítása ezekre a koordinátákra épp x.*

Az eddig bevezetett fogalmakkal készen állunk Shannon 1948-ban megjelent egyik alaptételének kimondására[1].

6. Tétel. *A forrásra és a csatornára tett feltevéseink mellett minden csatornához létezik egy sok esetben számolható C Shannon-kapacitásnak nevezett szám, amelynek értéke csak a csatornától függ, és amely rendelkezik a következő tulajdonsággal. Tetszőleges rögzített ε-hoz és δ-hoz találunk olyan C-nél legfeljebb δ-val kisebb rátájú blokk kódot, és egy hozzá tartozó dekódoló algoritmust, amelyekkel a hibavalószínűség ε-nál kisebb. Továbbá teljesül, minden C-nél nagyobb D számra, hogy tetszőleges olyan blokk kódsorozatra, amelyhez tartozó $\frac{k}{n}$ rátasorozat D-hez tart, úgy, hogy $k, n \to \infty$, igaz, hogy a kódsorozathoz tartozó hibasorozat nem tart 0-hoz.*

[1] pontosabban speciális esetének ismertetésére

A két példának hozott csatorna esetén a kapacitás csökkenő függvénye a p-nek és a σ-nak, vagyis minél nagyobb a zavaró hatás, annál több csatornán átküldött karakterre van szükség a forrás kis hiba melletti továbbítására. A BSC csatorna zaja ellen a legegyszerűbb védekezési technika, ha minden forrásbetűt előre rögzített c-szer küldünk át a csatornán, a felhasználó pedig megvárja az egy forrásbetűhöz tartozó c kimenetet, és a forrásbetűt a többször szereplő kimenettel azonosítja. Ezzel a technikával tetszőlegesen kis hibavalószínűséget el lehet érni c tetszőleges nagyra növelésével, de ekkor a kódolás-dekódolás $\frac{1}{c}$ rátája 0-hoz tart. A fent ismertetett tétel azt állítja, hogy ez az eljárás nem az optimális, hiszen ugyan csak a kapacitás alatt, de lehetséges pozitív rátával tetszőleges kis hibavalószínűség mellett üzenetet továbbítani. Sajnos a bizonyításban a kódoló leképezés egy véletlen halmazból kerül kiválasztásra, így nem tudni melyik kód bizonyult jónak. Másrészt a tételben a kódhoz tartozó dekódoló gyakorlatban kivitelezhetetlen számolási nehézségekhez vezet. Így a tétel 1948-as megjelenését követően alapvető kérdéssé vált a csatornákhoz olyan kódoló-dekódoló pár tervezése, amely számolása a gyakorlatban kivitelezhető, és amely rátája a tételben szereplő határhoz közel van, megfelelően alacsony hibavalószínűség mellett. Mint említettem körülbelül 50 évnyi kutatás után, végül a gyakorlatban fontos csatornákon Turbó-kóddal és az LDPC kóddal sikerült elérni a kapacitást.

Megjegyzem, hogy nagyon gyakran a mérnökök megelégednek az átlagos betűnkénti hibavalószínűség minimalizálásával, vagyis $\frac{1}{k}\sum_{i=1}^{k} p(U_i \neq V_i)$ leszorításával. Az előző tétel bizonyításának módosításával belátható, hogy az ismertetett határ felett, ha ezzel a gyengébb kritériummal definiáljuk a hibavalószínűséget, akkor sem lehet aszimptotikusan elérni tetszőlegesen kis hibavalószínűséget. A kódok teljesítményét legtöbbször ennek empirikus változatával mérik. Szimulálják a kódolást, a csatornán való áthaladást, és a dekódolást, majd megnézik, hogy a kapott betűk hányad része egyezik az eredeti üzenet betűivel.

A dolgozatban szereplő dekódolási eljárások lényege a már ismert marginalizálás. Jelöljük a forrásblokk értékkészletét $\mathbf{u}$-val, a csatorna kimeneteként kapott blokkot $\mathbf{y}$-al. A dolgozatban részletesen szereplő minden kódoló-dekódoló eljárásnál a célunk a $p(u_i = 1|\mathbf{y})$ és $p(u_i = 0|\mathbf{y})$[2] marginális valószínűségek kiszámolása vagy közelítése. Végül a dekódoló a valószínűbb értékekkel tér vissza. Felhasználva a Bayes-formulát[3] adódik:

$$p(u_i = 1|\mathbf{y}) = \sum_{\mathbf{u}:u_i=1} p(\mathbf{u}|\mathbf{y}) = \frac{\sum_{\mathbf{u}:u_i=1} w(\mathbf{y}|h(\mathbf{u}))p(\mathbf{u})}{g(\mathbf{y})}, \tag{5.2}$$

ahol h a kódolót definiáló függvény, $g(\mathbf{y})$ pedig az $\mathbf{y}$ többdimenziós tömegeloszlása vagy sűrűségfüggvénye. Fontos, hogy $g(\mathbf{y})$ konstansnak tekinthető, továbbá $p(\mathbf{u})$ egyenletes eloszlású. A különböző kódoló eljárásoknak az a lényege, hogy $h(\mathbf{u})$ különböző faktorizációjához vezetnek, amelyeken a Belief Propagation algoritmus hatékonyan tud működni. Megjegyzem, hogy a betűnkénti marginális valószínűség kiszámolása helyett kereshetnénk 4.3 rész mintájára a maximális konfigurációt

[2]Megjegyzem, hogy folytonos kimenetű csatorna esetén a feltételben 0 valószínűségű esemény van. Ekkor a feltétel zsugorodó környezetének határértékként kapjuk meg a feltételes valószínűségeket.

[3]A következő képletben folytonos és diszkrét változók együtt szerepelhetnek, határértékkel látható, hogy helyes a formula.

is. Említés szintjén szóba kerül majd az ezt számoló Viterbi algoritmus. A következő kódtípus az LDPC kód alapja.

24. Definíció. *Azt mondjuk, hogy C egy bináris lineáris (n,k) kód, ha C k dimenziós lineáris altér $\{0,1\}^n$-ben. A kódoló leképezést definiálhatjuk úgy, hogy egy G mátrix soraiba elhelyezzük a C altér egy bázisát. Azt mondjuk, hogy az $x \in \{0,1\}^k$ sorvektor kódolt megfelelője az $xG \in \{0,1\}^n$ vektor. Igaz az, hogy minden lineáris kódot megadhatunk H $(n-k) \times n$-es ellenőrző mátrixal, amelyre $x \in C$ akkor és csak akkor, ha $xH^\top = (0,\dots,0)$. Felhívom a figyelmet arra, hogy a lineáris kód speciális blokk kód.*

7. Tétel. *Tetszőleges C (n,k) paraméterű lineáris kódra teljesül, hogy találunk olyan G generátormátrixot, amely pont a C kódot határozza meg, úgy, hogy a G által definiált kódolás szisztematikus.*

A továbbiakban felteszem, hogy ha lineáris kódról van szó, akkor azt szisztematikusan kódoljuk. Ennek a feltevésnek fontos szerepe lesz a lineáris kód információelméleti felhasználásában.

A lineáris kód ellenőrző mátrixos megadása elvezet minket a Tanner-Gráf fogalmához, amely Tanner [5] dolgozatából származik. Az $xH^\top = (0,\dots,0)$ egyenletrendszer $n-k$ darab kényszert jelent, amit a kódszavaknak teljesíteniük kell. Így nézve a kódba tartozás karakterisztikus függvénye előáll lokális kódok karakterisztikus függvényeinek szorzataként:

$$1_C(x_1,\dots,x_n) = 1_{C_1}(X_{C_1}) \cdots 1_{C_{n-k}}(X_{C_{n-k}}), \tag{5.3}$$

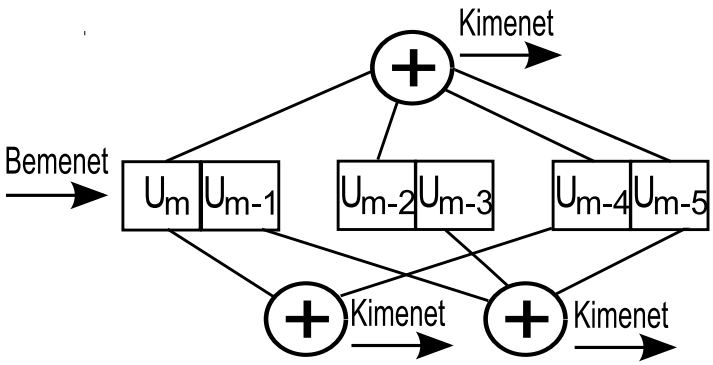
ahol a C_i kód a $H^\top$ mátrix i. oszlopa által definiált kód, vagyis azon vektorok tartoznak bele, amelyek skaláris szorzata a definiáló vektorral 0. X_{C_1} azon koordináták gyűjtőjelölése, amelyeken a definiáló vektor oszlopában 1-s áll. Az 5.3 faktorizációhoz tartozó Faktor-Gráf a Tanner-Gráf.

Most a Turbó-kód eredeti konstrukciójában kulcsszerepet játszó konvolúciós kódokról szeretnék írni. Több megközelítéssel lehet ismertetni ezt a kódtípust, én a léptető-regiszteres[4] ismertetés mellett döntöttem.

25. Definíció. *Az (n,k,m) paraméterű konvolúciós kód alatt, ahol n-el a kimenet betűszámát, k-val a bemenet betűszámát, m-el pedig a kódoló k betű tárolására alkalmas memóriacelláinak a számát jelöljük, a következőt értjük. Kezdetben minden memóriacella csupa nullából álló vektorral van feltöltve. A kódoló a blokk kódolóhoz hasonlóan megvár k forrásbetűt. Ezt követően egy előre rögzített szabályt követve összeadogat a kételemű testben a friss bemenet és a memóriacellák betűi közül néhányat, a kapott érték lesz az első kimenet. Összesen n darab összeadási szabályt követve adódik az n betűből álló kimenet. A folyamat végeztével minden memóriacella tartalma a következő memóriacellába másolódik (utolsó tartalma törlődik), majd várjuk az új bemenetet. Az 5.2 ábra tanulmányozásával válik igazán világossá a definíció.*

Megjegyzem, hogy a konvolúciós elnevezés egy másik megközelítésből ered, amikor is a kódolót polinomokkal való konvolúcióval definiáljuk. Vegyük észre, hogy általában a konvolúciós kódok

[4] angol szakirodalomban shift-register

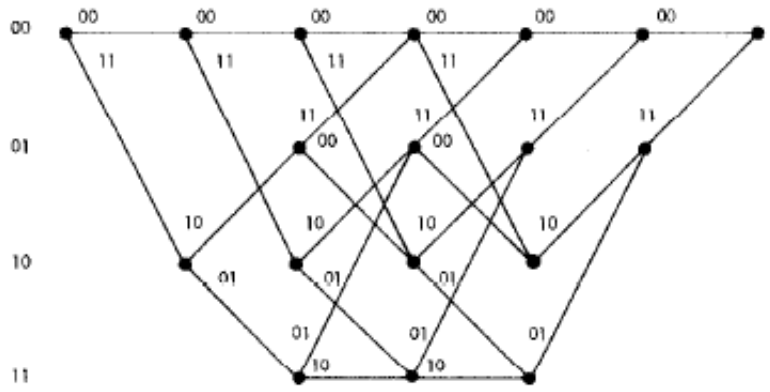


5.2. ábra. (3,2,2) paraméterű konvolúciós kód léptető-regiszteres ábrázolása

nem blokk kódok, hiszen a ugyanahhoz a bemenethez a memória tartalmától függően más kimenet tartozik. A gyakorlatban azonban mégis blokk kódként gondolhatunk rá a következő levágó eljárás segítségével. Választunk egy L számot, és azt mondjuk, hogy kL betűt kódolunk. Miután átküldtük a kL betűt, annak érdekében, hogy a memóriacellák tartalma nullelemekből álljon, definíció szerint átküldünk még km nullelemet, így a kódolás összességében $n(m+L)$ betűt eredményez, vagyis ezzel a kiegészítéssel egy $(n(m+L), kL)$ blokk kódhoz jutunk. Tipikusan L sokkal nagyobb, mint m, így ennek a blokk kódnak a rátája közel $\frac{k}{n}$. A továbbiakban konvolúciós kód alatt mindig levágott konvolúciós kódot értek, a hozzá kapcsolódott paraméterjelöléseket megtartom. Megjegyzem, hogy a blokk kóddá alakított konvolúciós kód is lehet szisztematikus.

Most a konvolúciós kódok nagyon fontos ábrázolásáról a trellis gráfról írok. A memóriacellák összes lehetséges állapota 2^{km}. Vegyünk fel egy $2^{km} \times (L+m+1)$-as négyzetrácsot. A négyzetrács csúcspontjai lesznek a trellis gráf potenciális csúcsai. A sorok a memóriacellák lehetséges állapotát, az oszlopok az iterációs lépéseket jelölik. Nevezzük a legelső oszlopot 0. oszlopnak. A 0. oszlopban tartsuk meg a memóriacellák induló, vagyis az azonos nullelem tartalomnak megfelelő állapothoz tartozó csúcsot, a többit töröljük az oszlopban. Tudjuk, hogy 2^k lehetséges bemenetet kaphat a kódoló az első iterációkor. Kössük össze a 0. oszlopban megmaradt csúcsot azokkal az első oszlopbeli csúcsokkal, amelyekhez tartozó memóriaállapot kialakulhat megfelelő bemenettel belőle. Írjuk rá az élre, hogy az első iterációs lépés során az él behúzásáért felelős bemenet milyen kimenetet eredményez. Az 1. oszlop azon csúcsait töröljük, amelyekbe nem vezet él a 0. szint egyetlen megmaradt csúcsából. Ha az i. oszlopig kész vagyunk, akkor ugyanezt csináljuk. Összekötjük az i. oszlopban megmaradt csúcsokat azokkal az $i+1$. oszlopban levő csúcsokkal, amelyekhez tartozó állapot elérhető a vizsgált csúcsnak megfelelő állapotból. Az éleket megfelelően címkézzük, majd a bejövőél nélküli csúcsokat töröljük. Felhívom a figyelmet arra, hogy levágott konvolúciós kóddal dolgozunk, így az L. szint után minden csúcsból, csak egy, következő szinten levő csúcs érhető el, mégpedig az, amelyik az azonosan 0 bemenet miatt alakulhat ki. Tekintve, hogy pont annyi azonosan 0 blokkot küldünk át az eljárás végén, amennyi memóriacella van, az utolsó oszlopban egyetlen csúcs marad, az azonosan nullelemből álló állapotnak megfelelő csúcs. A kapott gráfnak az a jó tulajdonsága, hogy minden az egyetlen 0. oszlopbeli csúcsból az egyetlen $L+m$. oszlopbeli csúcsba menő $L+m$ élből álló útnak megfelel egy lehetséges kódszó, továbbá a kódszót kiolvashatjuk az út címkéiből.

Egy $(2, 1, 2)$ $L = 4$ bináris konvolúciós kódhoz tartozó trellis gráfot mutat be az 5.3 ábra.



5.3. ábra. [3]-ból származó ábra: $(2, 1, 2)$ $L = 4$ konvolúciós kódhoz tartozó trellis gráf. Baloldalon látható, hogy a soroknak milyen állapotok felelnek meg.

5.2. Előre-Hátra algoritmus

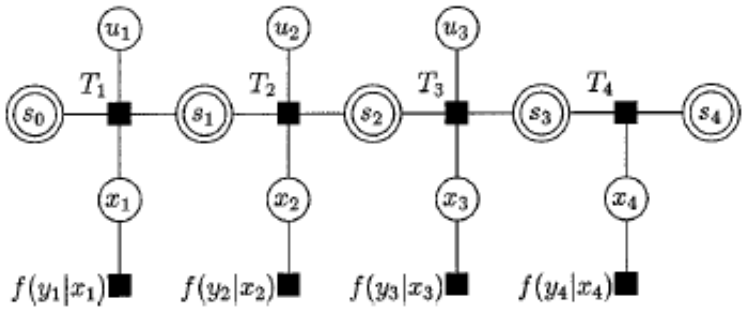
Az Előre-Hátra algoritmust BCJR algoritmusnak[5] is hívják a [3] cikk szerzői után. Ebben az 1974-ben megjelent cikkben az Előre-Hátra elnevezés nem szerepel, de mint látni fogjuk az algoritmus szerkezete sugallja az elnevezést. Az említett cikkben az algoritmus rejtett Markov modellen dolgozik, a kódolási alkalmazás speciális esetnek tekinthető. A részben bemutatom, hogy a hagyományos valós műveletek melletti Sum-Product algoritmus rejtett Markov modellen futtatva épp a [3] cikkben található algoritmushoz vezet. Ezt követően megmutatom hogyan lehet az algoritmust a konvolúciós kódok trellis gráfjához köthető dekódolásra használni. Ezt követően [19] alapján szólok néhány szót különböző általánosításokról, többek között lineáris kódok dekódolásáról. Végül szó lesz farok nélküli trellis[6] reprezentációról, amely fontos gyakorlati alkalmazása a 4.2 részben tárgyalt egy körös reprezentációnak. A rész megírásához [18]-t és [32]-t is felhasználtam.

Általános megközelítésben[7] az olyan valószínűségi modelleket nevezzük rejtett Markov modellnek, amelyekhez tartozó Faktor-Gráf az 5.4 ábrának megfelelő alakú. Kicsit pontosabban adott egy s_i-vel jelölt Markov-lánc, amely átmenetmátrixa függhet az u_i paramétertől, amely maga is valószínűségi változó. Az u_i-nak a Turbó-kód dekódolásában lesz fontos szerepe. Továbbá a Markov-lánc szomszédos állapotainak adott egy x_i függvénye, amely zajos csatornán halad át, és a végeredményként kapott y_i változót megfigyeltnek tekintjük. A megfigyelést jelen esetben behelyettesítéssel kezeltük. A legklasszikusabb rejtett Markov-lánc úgy néz ki, hogy nincsenek a modellben az u_i változók (vagy triviálisak), az x_i pedig megegyezik s_i-vel. Így a modell egy Markov-láncból áll, amelyet nem közvetlenül, hanem egy csatornán keresztül figyelhetünk csak meg.

[5]Megjegyzem, hogy a valamivel hamarabb kitalált Baum-Welch algoritmus is szorosan kapcsolódik a tárgykörhöz, a dolgozatban ezzel részletesebben nem foglalkozom.

[6]az angol szakirodalomban tail-biting trellis

[7]ez a megközelítés nem teljesen konvencionális



5.4. ábra. [18]-ból származó ábra. Általános rejtett Markov modellhez tartozó Faktor-Gráf. Az u_4 hiánya nem szükségszerű, arra illusztráció, hogy maradhatnak ki a modellből változók.

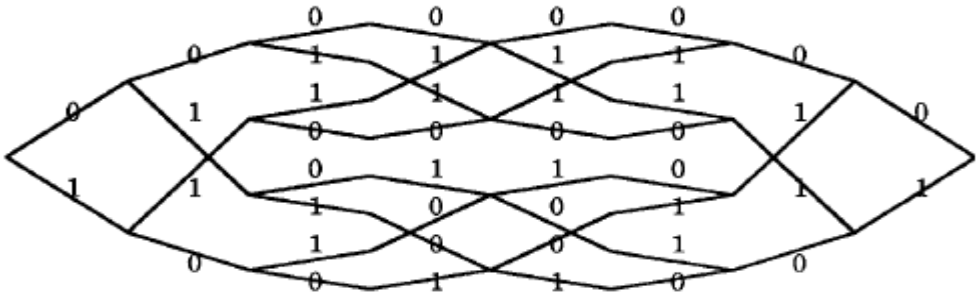
Az elsődleges cél a modell marginális valószínűségeinek meghatározása. Miután az 5.4 ábrának megfelelő Faktor-Gráf fa, ezért ez pontosan megtehető a Sum-Product algoritmus fán definiált, hagyományos valós műveletekkel működő változatával. Nézzük meg mindezt [3] cikk szempontjából. A cikkben egy kicsit speciálisabb modellel dolgoznak. Nem szerepelnek az u_i változók a modellben, továbbá az s_i változók alkotta Markov-lánc kezdő és végállapotát ismertnek és ugyanannak tételezik fel. Mindezt a megszokott módon, az 5.4 ábrán a két szélső változócsúcshoz kapcsolt egyfokú függvénycsúcsok felvételével modellezhetjük. Az 5.4 ábrán a bekezdésben említett speciális feltételekkel futtatva a Sum-Product algoritmus fán definiált változatát, hagyományos valós műveletek mellett, kiegészítve a két szélső csúcshoz kapcsolódó egyfokú függvényekkel, pont megkapjuk [3]-ban szereplő algoritmust. Az algoritmust rögtön Sum-Product szempontjából ismertetem. Két iterációs lépés után a szélső s_i csúcsok megkapnak minden szükséges információt ahhoz, hogy üzenjenek a szomszédoknak. Ezen üzenetek továbbítása után az üzeneteket megkapó csúcsok is készek üzenni. Így felhasználva, hogy egy élen bejövő üzenet nem befolyásolja a kimenőt, adódik egy szép kép. Ha α_i-vel jelölöm a láncon balról jobbra küldött üzeneteket, β_i-vel a jobbról balra küldött üzeneteket, akkor ezek rekurzívan végigáramlanak a láncon egymástól függetlenül. Innen adódik az Előre-Hátra elnevezés. Miután minden láncbeli csúcs megkapta a neki szánt α_i és β_i vektorokat, koordinátánkénti szorzatukból ki tudják számolni a saját marginális valószínűségüket. [3] a modellbe épített függetlenségi állításokat felhasználva jutott a Sum-Product algoritmus által egyszerűen biztosított rekurziókra. Megjegyzem, hogy algoritmust adtak két, a láncban egymást követő változócsúcs, közös marginális függvényeinek kiszámolására is. Ez az eredmény is adódik a Sum-Product algoritmusból, úgy, hogy a 3. fejezet 4. Állítását felhasználva meghatározzuk s_{i-1}, s_i, x_i közös marginális függvényét, majd a kapott függvényből az x_i változót szummázással eltüntetjük.

Most megmutatom, hogyan lehet trellis gráf segítségével dekódolni. Minden trellis gráfhoz rendelhetünk egy rejtett Markov modellt. Az s_i változók a trellis gráf oszlopainak felelnek meg, vagyis a nekik megfelelő valószínűségi változók értékkészlete a konvolúciós kód memóriacellájának tartalma. A 0. és az utolsó oszlophoz tartozó változók csak az azonosan nulla memóriatartalomnak megfelelő értéket vehetik fel. A Markov-lánc átmenetmátrixát úgy szerkesztjük meg, hogy a trellis gráf élei mentén lehetséges az átmenet egyenletes valószínűséggel a forrásról tett kezdeti feltételeinkkel összhangban. A lánchoz kapcsolódó x_i változó pedig az élekre írt címkéknek, vagyis a

kódoló kimeneteinek felel meg, így determinisztikus függvénye az él két végpontjának megfelelő változóknak. Ilyen szereposztásban az Előre-Hátra algoritmussal ki tudjuk számolni az oszlopokhoz tartozó pontos marginális valószínűségeket. Mivel a memória tartalma a bemeneti betűkből áll, az állapotok marginális valószínűségeiből könnyű számolással adódnak a bemeneti bitek pontos marginális valószínűségei. Az algoritmus futási ideje rögzített memória méret esetén a konvolúciós blokk kód hosszától lineárisan függ. Azonban a szükséges számítási idő exponenciálisan növekszik a memória növelésével. [4]-ben találhatóak azt az intuíciót alátámasztó tények, hogy annál jobb a konvolúciós kód, minél nagyobb a memória. Így önmagában ezzel a konstrukcióval nem lehet elérni a Shannon-kapacitást. Megjegyzem, hogy a Viterbi algoritmus körülbelül ugyanígy működik, azzal a különbséggel, hogy Max-Product algoritmust használ, emiatt betűnkénti maximalizálás helyett a maximális valószínűségű blokkot adja meg teljesen pontosan. Az említett megjegyzés a Viterbi algoritmusra is igaz.

A Turbó-kód működésének megértéséhez fontos kiegészítést szeretnék tenni. Az ismertetett Előre-Hátra dekódolót módosíthatjuk úgy, hogy a forrást közvetlenül is bevesszük a modellbe (eddig a memória állapotain keresztül volt része a modellnek). A forrás betűinek feleltessük meg az az 5.4 ábrán szereplő u_i változókat. A memóriaállapotokhoz köthető Markov-lánc átmenetmátrixa függ az u_i változótól, ismerve az értékét minden állapotból csak egy másik állapotba lehetséges az átmenet. Ennek a bővítésnek az előnye az, hogy lehetőség van információt vinni a rendszerbe. Vagyis ha akarunk, akkor kapcsolhatunk az u_i eloszlását leíró egyváltozós függvényeket az u_i változókhoz. A Sum-Product algoritmus a bővített modellen is ugyanúgy dolgozik. Vegyük észre, hogy az 5.4 ábra alakja miatt, az u_i-ket mindenképpen függetlennek feltételezzük.

Lineáris kódok trellis reprezentációja alatt olyan címkézett élű gráfot ért a szakirodalom, amely a következő tulajdonsággal rendelkezik. Meg van jelölve egy kitüntetett kiinduló csúcs. A gráfot szintekre oszthatjuk aszerint, hogy minimálisan milyen hosszú úton érhetőek el a kiinduló csúcsból. Feltétel, hogy az utolsó szinten csak egy darab zárócsúcsnak nevezett csúcs található. Ezenkívül teljesül, hogy a kiinduló és a zárócsúcs között haladó minimális élszámú utak címkéi felelnek meg a kódszavaknak. Egy 8 hosszúságú lineáris kódhoz tartozó trellis gráfot mutat be az 5.5 ábra.



5.5. ábra. [19]-ből származó ábra. 8 hosszúságú lineáris kódhoz tartozó trellis

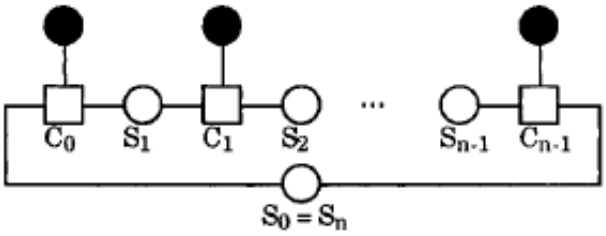
Lineáris kód trellis gráfjához is rendelhetünk rejtett Markov modellt. A trellis gráfot oszlopokra osztjuk, aszerint, hogy a csúcsok minimálisan milyen hosszú úton érhetők el a kiinduló csúcsból. A lánc változói az oszlopok lesznek. A változók értékkészletének a számossága az egyes oszlo-

pokban szereplő csúcsok számával egyezik meg. A két szélső oszlophoz tartozó változó a definíció miatt csak egy értéket vehet fel. Az x_i változókat most is a trellis gráf éleinek és címkéinek segítségével, determinisztikus függvénnyel definiálhatjuk. Ekkor a láncbeli csúcsok nem köthetőek memóriatartalomhoz, mesterséges változóknak hívhatjuk őket. A mesterséges változók marginális eloszlásainak meghatározása nem segít a dekódolásban. Azonban a Sum-Product algoritmussal az 5.4 ábrán látható x_i változók marginális függvényét is meg lehet határozni. Ezek a változók pedig a kódoló kimenetei, vagy más szóhasználattal a csatorna bemenetei. Kihasználva azt, hogy feltettem, minden lineáris kód szisztematikus, a szisztematikus helyeknek megfelelő x_i-k marginális valószínűségeinek meghatározásával dekódoló algoritmushoz jutunk.

Az előző bekezdés nem tárgyalta azt, hogy hogyan jutunk lineáris kódhoz tartozó trellis gráfhoz. Elmondható, hogy egy lineáris kódhoz több trellis reprezentáció is tartozhat. [3] cikk ad konstrukciót, amellyel tetszőleges lineáris kódhoz lehet trellis gráfot rajzolni. A dekódolási komplexitás minimalizálása érdekében érdemesebb a lehetőségek közül azt választani, amelyben a mesterséges változók értékkészlete a lehető legkisebb.

Az eddigieket úgy foglalhatnám össze, hogy kódok trellis reprezentációja segítségével fa Faktor-Gráfon vett marginális valószínűségek számolására vezettük vissza a dekódolást. A Faktor-Gráf függvénycsúcsai és mesterséges változói lokális kényszereket definiáltak. Ezek segítségével írtuk le a kódot. Ez a hozzáállás lehetőséget biztosít általánosításra. Forney [19]-ben részletesen foglalkozik lineáris kódok alacsony komplexitású Faktor-Gráf reprezentációjával. A cikkben találhatóak tételek, amelyek mutatják, hogy a kód reprezentációja annál gazdaságosabb a Sum-Product komplexitásának szempontjából, minél összefüggőbb a vizsgált gráf. Így fa alakú rejtett Markov modellel nem lehet igazán alacsony komplexitással kódolni. Tehát a kört is tartalmazó reprezentáció előnyösebb a Sum-Product algoritmus komplexitása szempontjából. De nem szabad elfelejteni, hogy kört is tartalmazó reprezentációnál a kapott marginális valószínűségek csak közelítései az igazinak, nincs jól feltérképezve a közelítés nagyságrendje. Mindazonáltal dekódolásnál, csak a legvalószínűbb állapotokra vagyunk kíváncsiak, így még ha a közelítés pontatlan is, nem biztos, hogy ez a tény gondot okoz. Felhívom a figyelmet a dolgozat a 3.15 és a 4.13 képleteire, és a körülöttük levő megjegyzésekre, továbbá a 4.3 részre, amelyek pont arra mutatnak példát, hogy a pontatlanság nem okoz minden esetben gondot.

A rész lezárásaképpen szeretnék megemlíteni egy farok nélküli trellis gráfnak nevezett konstrukciót, amely összekapcsolja a részt a 4.2 résszel, vagyis a pontosan egy kört tartalmazó reprezentáció vizsgálatával. A farok nélküli trellis gráfban is bijektív megfeleltetésben állnak egymással az első oszlop és az utolsó oszlop csúcsai. A gráfban azok az első oszlopból utolsó oszlopba menő utak címkéi felelnek meg kódszavaknak, amelyek ugyanolyan kezdőállapotból indulnak, mint amilyenbe érkeznek. Így voltaképpen körök felelnek meg kódszavaknak. Ezzel a megnövelt dimenzióval gazdaságosabban lehet leírni a kódot, így a dekódolási komplexitás csökkenhet. Az 5.6 ábra egy ilyen reprezentációhoz tartozó Faktor-Gráfot mutat be.



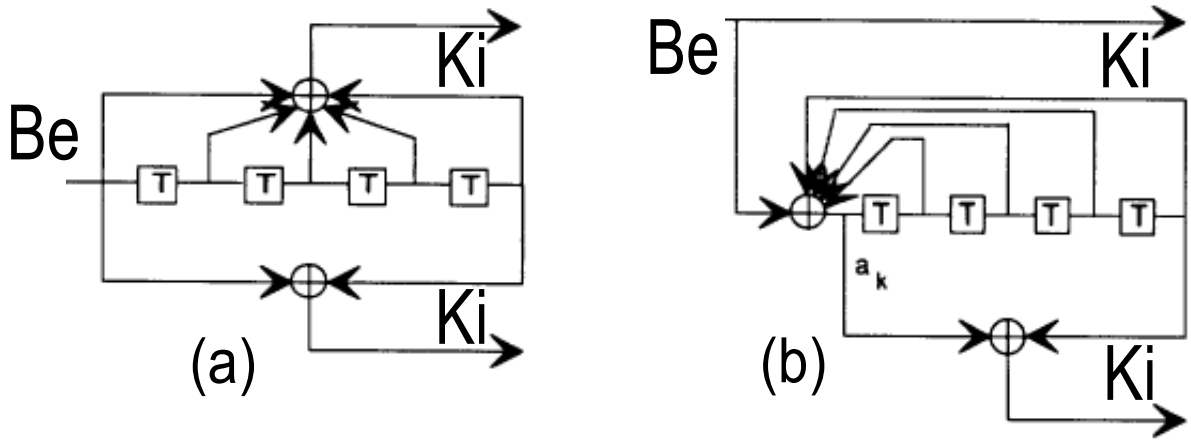
5.6. ábra. [19]-ből származó ábra. Farok nélküli trellis reprezentációhoz tartozó Faktor-Gráf. A beszínezett csúcsok felelnek meg a kódszó betűinek.

5.3. Turbó-kód

A részben vázolom a Turbó-kód felépítését, megmutatom, hogy hogyan lehet Belief Propagation algoritmus segítségével leírni, végül megjegyzek intuíciókat, amelyek jobban megértetik, hogy miért is működik a konstrukció. A Turbó-kód ismertetését [12] alapján teszem meg, néhol utalva az eredeti [7]-re is. A két leírás közötti eltérés a lényeget nem érinti. A rész megírásához felhasználtam [17], [18]-t is.

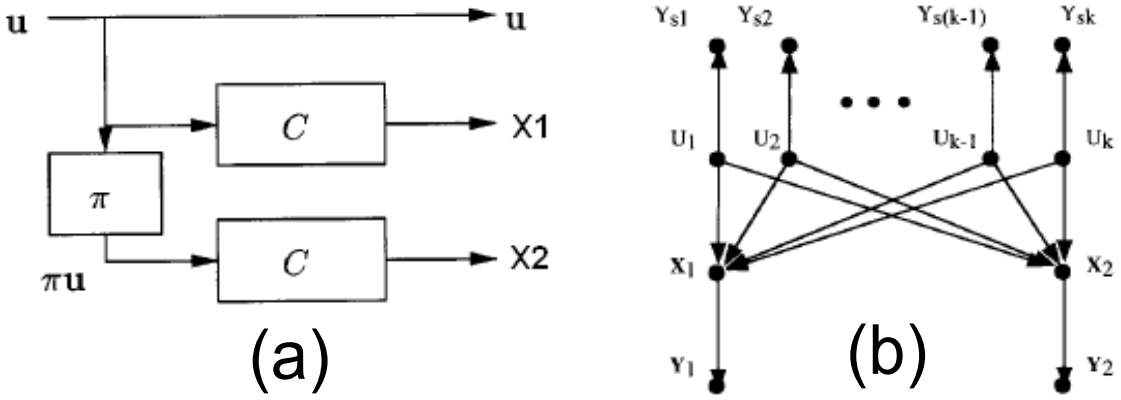
A Turbó-kód alapját az előző fejezetben definiált konvolúciós kód képezi. Pontosabban egy módosított változata, amelyet szisztematikus rekurzív konvolúciós kódnak nevezhetünk. A rekurzív szó arra utal, hogy a memóriákban tárolt állapotok nemcsak a korábbi bemenetektől függnek, hanem az előző kimenetektől is. Konvencionális mérnöki diagrammon mutatja be az 5.7 ábra a különbséget a korábban definiált konvolúciós kód, és a rekurzív kód között. A konvolúciós kódok paraméterjelölése a megszokott. Az ábrát úgy kell érteni, hogy minden iterációs lépésben az új bemenet és a memóriákban tárolt betűk elindulnak a nyilak mentén, ha többfelé ágazást tapasztalnak, akkor több úton is haladnak párhuzamosan, ha összeadó függvényhez érnek, akkor más állapotokkal összeadódva haladnak tovább az összeadó függvényből kifelé mutató nyilak mentén, teszik mindezt addig, amíg memóriacellához vagy kimenethez nem érnek. A rekurzív konvolúciós kód fogalma régóta ismert volt, mégis a dolgozat előtt meglehetősen keveset használták. Megjegyzem, hogy a hagyományos konvolúciós kódoknál megszokott módon készíthetünk blokk kódot a rekurzív konvolúciós kódból. Annyi a különbség, hogy az eljárás végén nem azonosan 0 bemenetet kell küldeni a kódolónak, hanem a memória tartalmától függő bemenetet. Mindez a bemeneti bitek nyomon követésével könnyen megoldható. Emiatt a rekurzív konvolúciós kódok Előre-Hátra dekódolása, pontosan úgy történik, mint a hagyományos konvolúciós kódok dekódolása.

[7]-ben bevezetett Turbó-kód kis közelítéssel két $(2, 1, 4)$ paraméterű a cikkben kitalált, permutációval kiegészített, párhuzamos összekötésén alapult. Permutációval kiegészített párhuzamos összeköttetés alatt azt értem, hogy a konvolúciós kódokat szokásos módon blokk kódként kezelve az egyik konvolúciós dekódoló megkapja a kódolandó blokkot bemenetként, míg a másik kódoló a blokk egy előre rögzített permutáció szerinti, permutált változatát kapja bemenetként. Jelöljük a két dekódoló szisztematikustól eltérő kimeneteiből álló blokkokat X_1-el, és X_2-vel, a kódolandó forrás blokkot pedig U-val. Ekkor a csatornán az (U, X_1, X_2) betűsorozat kerül átküldésre. Megjegyzem, hogy megfelelő, a permutációt is érintő eljárással elérhető, hogy mindkét kódoló az azonosan



5.7. ábra. [7]-ből származó ábrák. Az (a) ábra egy (2,1,4) konvolúciós kódot, (b) egy (2,1,4) rekurzív konvolúciós kódot ábrázol. Az ábrák mérnöki gyakorlatban megszokott szemléltetés szerint készültek. A főszövegben részletesen le van írva, hogyan kell az ábrákat pontosan érteni.

0 állapotba kerüljön. Az 5.8 ábra (a) része mutatja a párhuzamos összeköttetés lényegét. A Turbó-



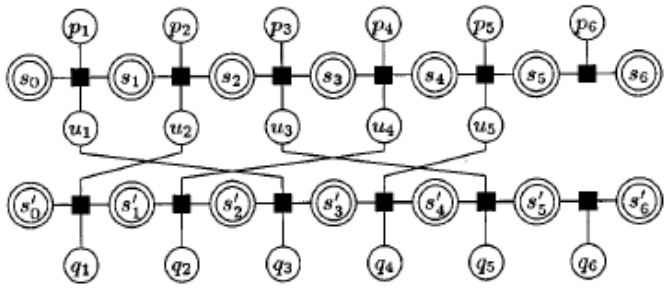
5.8. ábra. Az (a) ábra [18]-ból származik. Két kódoló permutációval kiegészített párhuzamos összeköttetését mutatja be. A (b) ábra [12]-ből származik. A Turbó-kód Bayes-háló megjelenését ábrázolja.

kód dekódolójának a lényege a két felhasznált rekurzív kód Előre-Hátra dekódolójának iteratív használata. A két dekódoló felváltva dolgozik. A kapott marginális függvényeket elküldik egymásnak, és a kapott információt az előző részben említett kiegészítéssel beépítik a dekódolásba. Kicsit pontosabban az első lépésben az első dekódoló az (U, X_1) bemenethez tartozó kimenetet felhasználva becslést ad U betűinek marginális valószínűségeire[8]. Ezt követően a számolást végző dekódoló elküldi az eredményt a másik dekódolónak a kódolásnál alkalmazott permutáció felhasználásával. A másik dekódoló az eredmény és az (U, X_2)-höz tartozó kimenet birtokában további becslést szolgáltat. A kapott eredményt aztán a kódolásnál használt permutációt használva visszaküldi az első dekódolónak, és így tovább egészen konvergenciáig. A pontos működést könnyen le lehet írni [12]-t

[8]Ezek a becslések a kapott információkhoz tartozó feltételes eloszlás pontos marginális függvényei.

követve Bayes-hálón. Az 5.8 ábra (b) része mutatja a modellhez tartozó Bayes-hálót. Az ábrán a forrás betűit külön változónak, míg a kód két nem szisztematikus részét egyesített csúccsal reprezentáljuk. Az Y_i változók a csatorna kimenetének betűi. Világos, hogy az ábrázolt Bayes-háló leírja a modellünket, a feltételes valószínűségeket leíró táblázatok is adódnak a modellből. Azt lehet mondani, hogy a kört is tartalmazó modellen futtatott Belief Propagation algoritmus, ha a frissítéseket a következő irreguláris ütemezés mellett használjuk: U csúcshalmazból kifutó élek, X_1-csúcshalmazból kifutóak, U-ból kifutóak, X_2-ből ..., épp a Turbó-kód dekódolásához vezet. A modell függvényei közé beépítettnek tekintjük a permutációt. Az X_i összevont változó U_i-k felé küldött üzeneteinek a kiszámolása a megszokott Pearl algoritmussal hosszadalmas lenne. A két rekurzív kód módosított dekódolói épp ezt a számolást végzik el gyorsan.

Összefoglalva a leírtakat a Turbó-kód [7]-ben ismertetett konstrukciója több új ötletet tartalmazott. Először is használta a kicsit elfeledett rekurzív konvolúciós kódot. Fontos tényező a permutációval ellátott párhuzamos kapcsolás. Továbbá az Előre-Hátra dekódoló algoritmus iteratív dekódolásra kialakított változata. A Turbó-kód sikerét későbbi szerzők pont az új eszközökben látják. Fontos, hogy a kapott összetett kód viselkedése nagyon közel van a véletlen kód viselkedéséhez. A véletlen jellegű permutációt biztosító eszköz[9] szerkesztése nem könnyű feladat. Mindazonáltal számos megoldás létezik. Látni fogjuk, hogy az LDPC kódnál is ez lesz a kulcs. Ha feltesszük, hogy sikerült véletlenül választanunk a kódolót, akkor a kapott kód nagy valószínűséggel olyan, hogy a legkisebb élszámú kör is hosszú. Így az algoritmus lokálisan pontos lesz, ami elég lesz a jó dekódoláshoz. Ugyanez a jelenség itt is felfedezhető. Ha az 5.8 ábra (b) részét kicsit részletesebb Faktor-Gráf nézőpontból nézzük, akkor a kapott 5.9 ábra gráfjára teljesül, hogy nincsenek kis körök a gráfban. Végezetül felhívnám a figyelmet arra, hogy a kódolási alkalmazásokhoz tartozó Faktor-Gráfokban rengeteg 0 található. Továbbá a Turbó-kód leírása alátámasztja az ütemezés fontos szerepét.



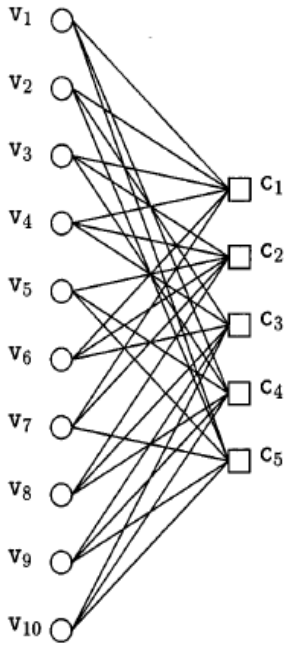
5.9. ábra. [18]-ból származó ábra. Az 5.8 ábra kicsit részletesebb Faktor-Gráf megfelelője. Itt most p és q jelöli x_1-t és x_2-t. A nekik megfelelő rejtett Markov-modell van részletesen kirajzolva.

[9] angol szakirodalomban interleaver eszköznek nevezik

5.4. LDPC kód

A részben Urbanke és társszerzői [20],[21],[22] cikkei alapján bemutatom a reguláris LDPC kódokhoz kötődő legfontosabb eredményeket. A kódhoz tartozó dekódoló Belief Propagation algoritmust használ.

26. Definíció. *Első közelítésben $C^n(d_v, d_c)$-el jelölöm azon lineáris kódoknak a halmazát, amelyek n hosszúak, és a kód Tanner-Gráfjában minden változócsúcs d_v fokú, a függvénycsúcsok pedig d_c fokúak. A definíció második felében ismertetett véletlen választás definiálja teljesen pontosan a halmazt. Ezeket a kódhalmazokat nevezzük reguláris LDPC kódnak. Megjegyzem, hogy a paraméterekből adódóan a függvénycsúcsok m száma épp $\frac{nd_v}{d_c}$. Ha ez nem egész, akkor üres a kódhalmaz. Az elkövetkezőkben felteszem, hogy nemüres kódhalmazzal dolgozunk. $C^n(d_v, d_c)$-ből egyszerű technikával választhatunk egyenletes valószínűség szerint egy kódot. A leendő n darab változócsúcsot rakjuk sorba, és mindegyikhez képzeljünk oda sorba rendezett d_v darab zsákot. Így sorba rendezve találunk nd_v darab zsákot, pont annyit amennyi élnek indulnia kell a változócsúcsoktól a függvénycsúcsok felé. Ugyanezt a sorba rendezést tegyük meg a függvénycsúcsoknál is, csak ott minden csúcshoz d_c zsákot vegyünk fel. Ezt követően az $nd_v = md_c$ elem permutációi közül egyenletes valószínűséggel válasszunk egy π permutációt. A változócsúcsok i. zsákját kössük össze a függvénycsúcsok $\pi(i)$. zsákjával. Ezt követően a zsákok közötti élekre tekintsünk úgy, mint a zsákoknak megfelelő csúcsok közti élekre. A randomizálási procedúra végén megkaptuk $C^n(d_v, d_c)$ egy egyenletes valószínűséggel választott elemét. Feltételezve, hogy az ellenőrző függvények függetlenek, a kód rátája $\frac{n-m}{n} = 1 - \frac{d_v}{d_c}$.*



5.10. ábra. [20]-ból származó ábra. Egy 10 hosszú (3,6) paraméterű LDPC kódot látunk az ábrán.

Megjegyzem, hogy a fenti definíció megenged csúcsok között párhuzamos éleket. Látni fogjuk, hogy ennek nem lesz jelentősége, ezeket az eseteket el fogjuk kerülni. Az a tény, hogy a Tanner-Gráf

fokszámai korlátosak, azt eredményezi, hogy a Belief Propagation algoritmuson alapuló dekódoló iterációs lépései gyorsan végezhetőek. Viszont a kódoláshoz generátormátrixra van szükség, amely nem feltétlenül ritka. Pont ez jelenti a fő különbséget a Turbó-kód és az LDPC kód között. Előbbi kódolása gyorsabban végezhető, míg utóbbi dekódolása sokkal inkább alkalmas a párhuzamosításra[10]. [22] foglalkozik LDPC kódhoz olyan szisztematikus kódolást eredményező generátormátrix adásával, amellyel a kódolás gyorsan végezhető. Mindez azt jelenti, hogy az adott kódolási algoritmus alapvetően kvadratikus a blokk méretben, de a kvadratikus alak konstansa kicsi. Így nagy blokkméret is praktikusan számolható.

[20]-ban az ismertetett dekódoló algoritmusok a Tanner-Gráfon működnek. Azért használtam többes számot, mert egy a Belief Propagation algoritmust is magába foglaló Massage Passing algoritmuscsalád keretein belül történik a vizsgálódás. Tekintve, hogy a gyakorlatban a legfontosabb a Belief Propagation dekódoló, továbbá a cikkben példának hozott minden, látszólag eltérő algoritmus is a Belief Propagation algoritmusból különböző diszkretizálásokkal megkapható, nem térek ki az általános definícióra, mindent átfordítok a Belief Propagation nyelvezetére.

A Faktor-Gráf az 5.10 ábra kiegészítése a csatorna kimenetet megtestesítő változócsúcsokhoz kapcsolt egyváltozós $w(y_i|x_i)$ függvényekkel. A cikkben a Loopy Belief Propagation működik speciális ütemezéssel. A kezdeti üzenetek az azonosan egy vektorok. Ezenkívül 0. iteráció is van, amikor is csak a függvénycsúcsok dolgoznak. Ez annyit jelent, hogy a 0. iterációban az egyváltozós csatornakimeneteknek megfelelő függvények elküldik üzeneteiket a változócsúcsokhoz. Így futtatva az algoritmust célunk a szisztematikus betűkhöz tartozó marginális valószínűségek közelítése a megszokott (4.3) képletet használva. Tippünk a forrás betűire a kapott közelítő marginális függvények valószínűbb értékei. Tekintve, hogy minden valószínűségi változó bináris, az üzenetek tekinthetőek az $\{1, -1\}$ halmaz feletti (p_1, p_{-1}) eloszlásoknak.

A legtöbb gyakorlati esetben feltehető, hogy az egyfokú függvénycsúcsok által küldött (p_1, p_{-1}) eloszlás szigorúan pozitív[11]. Ebből, tekintve, hogy a modell függvényei bináris lineáris kódok karakterisztikus függvényei, adódik, hogy az algoritmus futása során minden üzenet szigorúan pozitív lesz. Ebben az esetben az üzeneteket egyértelműen karakterizálhatjuk a $log\frac{p_1}{p_{-1}}$ log-likelihood hányadossal. Tegyük fel, hogy az eloszlás helyett ezek kerülnek átküldésre. Ekkor intuitív jelentést is társíthatunk az üzenet mellé, az üzenet előjele tipp az éllel kapcsolatban álló változó értékére, az abszolútértéke pedig a tippnek az erősségét jelentheti. A csatornáról tegyük fel a következő szimmetria tulajdonság:

$$w(y_t = q|x_t = 1) = w(y_t = -q|x_t = -1). \tag{5.4}$$

Az említett feltevés technikai könnyítést jelent. Következik belőle, hogy tetszőleges Faktor-Gráffal reprezentált, Belief Propagation algoritmussal dekódolt lineáris kódra igaz, hogy azon feltétel mellett, hogy a csatornán az $\mathbf{x}$ vektort küldtük át, a dekódolásunk átlagos betűnkénti hibája független az $\mathbf{x}$-től. Így a dekódoló hibájának vizsgálatához feltehető, hogy az azonosan 1 blokk volt a csatorna bemenete (csak olyan paraméterekkel foglalkozunk, amelyeknél az azonosan 1 vektor kódszó,

[10]Hiszen látjuk a az 5.8 ábrán, hogy a Turbó-kódnál két csúcs, az X_1 és az X_2 számolnak sokat.

[11]Ez a feltevés azért reális, mert bináris értékkészletű valószínűségi változókkal dolgozunk.

vagyis d_c-ről feltesszük, hogy páros). Ennek a bizonyítása technikai, megtalálható [20]-ban. Az érthetőség kedvéért megjegyzem, hogy az átlagos betűnkénti hiba rögzített LDPC kód és rögzített csatorna bemenet esetén is valószínűségi változó, az értéke a csatorna kimenetelétől függ, az ezt követő dekódolási folyamat a csatorna kimenetnek egy determinisztikus függvénye.

Rögzítsünk a $C^n(d_v, d_c)$-nek egy elemét. Tegyük fel, hogy az azonosan 1 blokkot küldtük át a csatornán. Továbbá tegyük fel, hogy a kód Faktor-Gráfjában nem található $2l$ vagy annál rövidebb kör[12], továbbá azt, hogy nem alakult ki párhuzamos él. A feltevések miatt a dekódolás teljesen szimmetrikus a $k \leq l$ iterációs lépésig. Így az l. iterációs lépésig minden változócsúcsból függvénycsúcsba mutató irányított él mentén minden iterációban ugyanaz az üzenet eloszlása. Hasonló állítás fogalmazható meg, a függvénycsúcsból változócsúcsba mutató élekre is. Figyelem, az üzenetek maguk is eloszlások, az eloszlásokat jellemző log-likelihood érték eloszlásának fejlődését fogjuk megfigyelni. Tekintve, hogy minden változónak korlátos d_v szomszédja van, ezért ha a függvénycsúcsból változócsúcsba menő irányított éleken haladó üzenetek eloszlása az l. iteráció végén olyan, hogy negatív érték felvételének valószínűsége kicsi, akkor a dekódolás átlagos betűnkénti hibavalószínűsége is kicsi lesz. A következőkben vázolom, hogyan tudjuk nyomon követni az l. iterációs lépésig az éleken haladó üzenetek eloszlását.

Tegyük fel, hogy a v_i változócsúcsból a c_i függvénycsúcsba mutató (v_i, c_i) irányított élen haladó log-likelihood üzenet eloszlását szeretnénk nyomon követni. Ehhez először vegyük észre, hogy ha $m_1^k, \ldots, m_{d_v-1}^k$-el jelöljük a k. iteráció során a v_i-be nem c_i-ből befutó log-likelihood üzeneteket, továbbá m_0-al jelöljük a v_i-hez kapcsolódó egyfokú függvénycsúcsból érkező időben konstans üzenetet, akkor a k. iteráció során a v_i-ből c_i-be menő log-likelihood üzenetet a következő módon számolhatjuk ki:

$$m_{v_i \to c_i}^k = m_o + \sum_{i=1}^{d_v-1} m_i^k. \tag{5.5}$$

Mint korábban leírtam a véletlent jelenleg csak az adja, hogy a csatorna bemenetén szereplő 1-esek milyen kimenetet eredményeznek. Így a 0. iterációban az éleken haladó log-likelihood üzenetekhez tartozó valószínűségi változók teljesen függetlenek. Tekintve, hogy nincs $2l$ vagy annál rövidebb kör a gráfban, az (5.5) képletben $k \leq l$ esetén független valószínűségi változókat adunk össze. A szimmetria és körmentesség miatt minden bejövő m_i^k üzenet eloszlása azonos P eloszlású. Így konvolúcióval $m_{v_i \to c_i}^k$ eloszlása megkapható a P-ből. Ez a feladat, ha az eloszlás diszkrét, akkor gyorsan megoldható a gyors Fourier transzformációval. Folytonos csatorna esetén az eloszlások diszkretizálásával tehetjük meg ugyanezt.

Az előző bekezdés mintájára sokkal több számolással megmutatható, hogy numerikusan ki tudjuk számolni a függvénycsúcsból kimenő üzenetek eloszlását a bemenő üzenetek eloszlásából. Így tekintve, hogy a csatorna átmenetvalószínűségeiből adódik a kezdeti üzenetek eloszlása, $2l + 1$ hosszúságnál rövidebb körök hiánya esetén az éleken haladó üzenetek eloszlása numerikusan kezelhető. Ezt az eljárást nevezzük sűrűség-fejlődésnek[13]. Jelöljük az eljárással kapott függvénycsúcsból

[12] kör hosszán az éleinek a számát értem

[13] angol szakirodalomban density evolution

változócsúcsba mutató élek l. iterációs eloszlását P^l-el. Vegyük észre, hogy P^l kiszámolásánál a kód n blokkméretének csak annyi a jelentősége, hogy elég nagynak kell lennie ahhoz, hogy lehetővé váljon, nincsenek $2l$ vagy annál kisebb körök a gráfban. Így P^l csak d_v és d_c függvénye. Fontos a már említett tény, hogy felhasználva a korlátos fokszámot, ha P^l negatív értékekre helyezett súlya kicsi, akkor lokális körmentességet feltételezve az átlagos betűnkénti hibavalószínűség is kicsi. Numerikus számolással meg lehet vizsgálni, hogy ha l és vele együtt n végtelenhez tart, akkor P^l eloszlásban a negatív értékek összvalószínűsége 0-hoz tart-e. Néhány bekezdéssel később belátom, hogy elég nagy n esetén $C^n(d_v, d_c)$-ből a rész elején definiált módon választva egy elemet a kapott kód hibás üzeneteinek a száma a hibás üzenetek[14] várható értékére koncentrálódik. A hibás üzenetek számának várható értéke pedig a körmentes esetben tapasztalt várható értékhez konvergál. Így ha azt tapasztaljuk, hogy P^l eloszlásban a negatív értékek összvalószínűsége 0-hoz tart, akkor az azt jelenti, hogy a csatornán megbízhatóan tudunk kódolni-dekódolni LDPC kóddal.

Az előző bekezdést továbbgondolva tegyük fel, hogy adott diszkrét csatornák egy valós α paraméterezett sorozata, amelyre teljesül, hogy ha az α_1-nek megfelelő csatornán lokális körmentességet feltételezve aszimptotikusan 0-hoz tart az átlagos betűnkénti hibavalószínűség, akkor $\alpha_2 < \alpha_1$ esetén is. [20] részletesebben foglalkozik ezzel a kérdéssel, én annyit említek meg, hogy BSC csatornák a p hibavalószínűséggel paraméterezve, továbbá diszkrét Gauss csatornák a σ szórással paraméterezve teljesítik a feltételt. Ilyen csatornasorozatoknál rögzített (d_v, d_c) mellett numerikusan lehet közelíteni azon értékek szuprérumát, amelyeknél aszimptotikusan 0-hoz tart az átlagos betűnkénti hibavalószínűség. Az említett két példánál maradva, így adódik egy p és egy σ érték, amelyek mellett lehetséges (d_v, d_c) paraméterű kódokkal tetszőleges kis hibával üzenetet továbbítani. Továbbá meghatározhatjuk ennek a 6. Tétel által biztosított elméleti maximumát az $1 - \frac{d_v}{d_c}$ kódrátából. A két érték közötti különbség a Shannon-kapacitástól való eltérés. Vagyis a klasszikus hozzáállástól eltérően nem konkrét csatornához kerestünk jól működő LDPC kódot, hanem rögzített rátájú LDPC kódhoz kerestük meg azt a csatornát, amin még képes nagy n esetén megbízhatóan információt továbbítani a kód.

A következőkben ki fogom mondani és be fogom bizonyítani a megígért koncentrációs tételt. Előtte ismertetek egy nevezetes tételt, kimondok egy fontos definíciót és bizonyítok egy lemmát. Fontos megjegyeznem mindezek előtt, hogy a tételben a változócsúcsokból a függvénycsúcsokba menő üzeneteket fogom vizsgálni. A frissítési szabályok megvizsgálásából adódik, hogy ha Q^l-el jelölöm lokális körmentességet feltételezve a változócsúcsokból a függvénycsúcsokba menő élek eloszlását, akkor könnyű látni, hogy P^l negatív félegyenesre helyezett súlya akkor és csak akkor tart 0-hoz, ha Q^l negatív félegyenesre helyezett súlya 0-hoz tart.

8. Tétel. *Azuma egyenlőtlenség: Legyen $Z_0, Z_1, \ldots$ Martingál, amelyre teljesül $\forall k \geq 1$-re:*

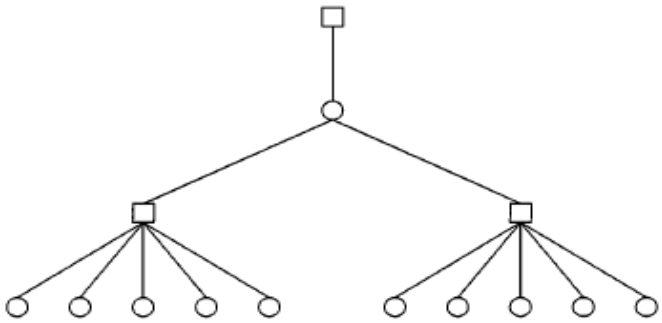
$$|Z_k - Z_{k-1}| \leq \alpha_k. \tag{5.6}$$

Ekkor $\forall l \geq 1$-re és $\forall \lambda > 0$-ra:

$$p(|Z_l - Z_0| > \lambda) \geq 2e^{-\frac{\lambda^2}{2\sum_{k=1}^{l} \alpha_k^2}}. \tag{5.7}$$

[14] Hibás üzenet alatt negatív üzenetet értek.

27. Definíció. *A G Faktor-Gráfban egy v-ből c-be mutató (v,c) irányított él l mélységű $N^l_{(v,c)}$ környezetén a $V^l_{(v,c)}$-beli csúcsok által G-ben kifeszített részgráfot értjük, ahol $V^l_{(v,c)}$ csúcshalmaz legyen a v-ből legfeljebb l hosszú olyan irányítatlan úton elérhető csúcsok halmaza, amely utak nem mennek át a (v,c) élen. Egy $d_v = 3, d_c = 6$ paraméterű LDPC kód (v,c) élének $N^2_{(v,c)}$ környezetét mutatja az 5.11 ábra.*



5.11. ábra. [20]-ból származó ábra. Egy $d_v = 3, d_c = 6$ paraméterű LDPC kód (v,c) élének $N^2_{(v,c)}$ környezetét mutatja

Fontos, hogy a (v,c) irányított élen haladó üzenet az l. iterációig csak az $N^{2l}_{(v,c)}$ részgráftól függ. Ha azt mondom, hogy $N^l_{(v,c)}$ fa, akkor abba beleértem, hogy nincs párhuzamos éle. A következő lemma állításának pontos megértéséhez érdemes másképp gondolni $C^n(d_v,d_c)$-ből való randomizálásra. A változócsúcsokhoz tartozó zsákok és a függvénycsúcsokhoz tartozó zsákok közötti $nd_v = md_c$ éleket húztunk be úgy, hogy az egyik végpontot rögzítettük. A randomizálás ugyanaz, ha mindezt mohón csináljunk. Látunk egy zsákot valamelyik oldalon, amelyben még nincs él, ekkor választunk neki a túloldalról a még üres zsákok közül egyenletes valószínűség szerint szomszédot. Ezt addig folytatjuk amíg üres zsákot látunk. Ez az eljárás az egyenletesen választott π permutáció randomizálását részletezi.

2. Lemma. *Válasszunk véletlenül egy kódot $C^n(d_v,d_c)$-ből. Rögzítsünk egy l^* iterációs számot. Legyen (v,c) egy irányított él a Faktor-Gráfban. Ekkor létezik $\gamma(l^*,d_v,d_c)$[15] konstans, hogy:*

$$p(N^{2l^*}_{(v,c)} \textit{nem fa}) \leq \frac{\gamma}{n} \tag{5.8}$$

Bizonyítás: Annak a valószínűséget fogjuk alulbecsülni, hogy $N^{2l^*}_{(v,c)}$ fa. Ha $N^{2l^*}_{(v,c)}$ fa, akkor a fokszámokból és l^*-ból kiszámolható a benne található változócsúcsok M_{l^*} száma, és a benne található függvénycsúcsok C_{l^*} száma.

Legyen $l < l^*$, továbbá m-el szokásosan a függvénycsúcsok számát jelöljük. Megvizsgáljuk, hogy mi annak a valószínűsége azon feltétel mellett, hogy $N^{2l}_{(v,c)}$ fa, annak, hogy $N^{2l+1}_{(v,c)}$ fa lesz. Ezt úgy képzelhetjük, hogy feltesszük, hogy $N^{2l}_{(v,c)}$ éleinek megfelelő éleket húztuk be először a zsákok között. Fontos, hogy ezek az $N^{2l}_{(v,c)}$-beli behúzott élek a $N^{2l}_{(v,c)}$ fa csúcsaihoz tartozó zsákokat kitöltik, kivéve a legalsó szinten levő csúcsokat. Így a feltett kérdés az, hogy a legalsó szinten levő

[15] így jelölöm a konstans paraméterektől való függését

csúcsok, jelen esetben paritási megfontolásokból változócsúcsok, zsákjaihoz milyen valószínűséggel társítunk függvénycsúcs zsákokat úgy, hogy a kiegészített objektum továbbra is fa marad. Ezt a gondolatot továbbgondolva, tegyük fel, hogy $N^{2l}_{(v,c)}$ fa, továbbá már sikerült k darab alsó szintbeli zsákhoz úgy élet társítani, hogy továbbra is fánk maradt. Ekkor annak a valószínűsége, hogy a következő zsákhoz olyan élet társítunk, amely a meglévő fán kívülre megy $\frac{(m-C_l-k)d_c}{md_c-C_ld_c-k}$. Hiszen összesen md_c zsák van a függvénycsúcs oldalon, ebből C_ld_c zsák az eddig behúzott élekkel telített, és még a frissen behúzott k darab él is eltelített k darab különböző függvénycsúcsokhoz eső zsákot. Továbbá a következő alsó szinthez kapcsolódó él behúzásakor akkor marad fa az objektumunk, ha a választott zsák nem tartozik egyik frissen kihúzott k csúcs zsákjai közé sem. Megjegyzem, hogy [20]-ban hibásan $md_c - C_l - k$ szerepel a nevezőben[16]. Szerencsére ez a kis hiba a lényeget nem érinti, mindkét mennyiséget ugyanúgy lehet elég nagy $m = \frac{nd_v}{d_c}$ esetén alulbecsülni:

$$\frac{(m-C_l-k)d_c}{md_c-C_ld_c-k} = 1 - \frac{kd_c-k}{md_c-C_ld_c-k} \geq 1 - \frac{k}{m} \geq 1 - \frac{C_{l^*}}{m}. \tag{5.9}$$

Ebből pedig iterációval következik, hogy:

$$p(N^{2l+1}_{(v,c)} \text{ fa} | N^{2l}_{(v,c)} \text{ fa}) \geq \left(1 - \frac{C_{l^*}}{m}\right)^{C_{l+1}-C_l}. \tag{5.10}$$

Teljesen hasonló becslés adódik a $2l+1$. környezetből a $2l+2$. szintre történő lépés esetén, egyszerűen a változócsúcsok és a függvénycsúcsok szerepe felcserélődik. A kapott eredményeket iteratívan felhasználva adódik a következő alsóbecslés:

$$p(N^{2l}_{(v,c)} \text{ fa}) \geq \left(1 - \frac{C_{l^*}}{m}\right)^{C_{l^*}} \left(1 - \frac{M_{l^*}}{m}\right)^{M_{l^*}}. \tag{5.11}$$

Ebből pedig komplementerrel és számolással adódik a lemma állítása. ■

9. Tétel. *[20] tétele: Mindent tegyünk fel, amit a részben feltettünk. $C^n(d_v, d_c)$-ből a definiált módon véletlenül választunk egy kódot, amivel tetszőleges forrásblokkot kódolunk. Rögzítsünk egy l dekódoló iterációs számot. Jelöljük Z-vel a Belief Propagation dekódoló l. iterációja során a változócsúcsból függvénycsúcsba menő hibás üzenetek számát*[17]*. Jelöljük p-vel a körmentes esetben kapott Q^l eloszlás negatív félegyenesre helyezett súlyát. Ekkor léteznek $\beta(l, d_v, d_c)$ és $\gamma(l, d_v, d_c)$ konstansok, amelyekre teljesül:*

(a) A hibás üzenetek száma a várható értéke köré koncentrálódik. Minden $\epsilon > 0$-ra:

$$p(|Z - E[Z]| > nd_v\frac{\epsilon}{2}) \leq 2e^{-\beta\epsilon^2 n}. \tag{5.12}$$

(b) A hibás üzenetek várható értéke a körmentes esetben tapasztalt várható értékhez konvergál. Minden $\epsilon > 0$ és $n > \frac{2\gamma}{\epsilon}$-ra:

$$|E[Z] - nd_v p| < nd_v\frac{\epsilon}{2}. \tag{5.13}$$

[16]Ez a hiba az elmélet szempontjából nem lényeges. A lemma azt az egyszerű állítást formalizálja, hogy rögzített l esetén elég nagy n-re nagy valószínűséggel a kapott Tanner-Gráfban nincsenek $2l$-nél rövidebb körök

[17]Abban az esetben, ha az azonosan 1 vektor volt a csatorna bemenete, a negatív üzenetek számát jelöli a valószínűségi változó.

(c) A hibák száma a körmentes esetben tapasztalt várható hibaszámra koncentrálódik. Minden $\epsilon > 0$ *és* $n > \frac{2\gamma}{\epsilon}$*-ra:*

$$p(|Z - nd_v p| > nd_v \epsilon) \leq 2e^{-\beta\epsilon^2 n}. \tag{5.14}$$

Bizonyítás: Első megjegyzésem, hogy a forrástól független a hibaszám, hiszen bármilyen kódszó is készül belőle, a kapott kódszóról hibavizsgálat szempontjából feltehető, hogy az azonosan 1 kódszóról van szó. Így az egész folyamatban két véletlen faktor van, a $C^n(d_v, d_c)$-ből való randomizálás, és a csatornából származó bizonytalanság. Előbbit egy nd_v hosszú permutáció, utóbbit a csupa 1 blokkhoz tartozó kimenet karakterizálja. Így Ω valószínűségi mezőre gondolhatunk úgy, mint az nd_v elemen vett permutációk terének és a lehetséges csatorna kimenetek terének direkt szorzatára. Ezek után lássuk az egyes pontok bizonyítását.

(c): Következik (a)-ból és (b)-ből.

(b): Vegyük észre, hogy sorszámozhatjuk az éleket aszerint, hogy a változócsúcs oldalon hanyadikok voltak a sorban. Legyen Z_i indikátor valószínűségi változó, amely értéke 1 ha az i. élen hibás az üzenet előjele. Ezt használva:

$$E[Z] = \sum_{i=1}^{nd_v} E[Z_i] = nd_v E[Z_1]. \tag{5.15}$$

Folytatva a gondolatmenetet:

$$E[Z_1] = E[Z_1|N_{e_1}^{2l} \text{ fa}] \cdot p(N_{e_1}^{2l} \text{ fa}) + E[Z_1|N_{e_1}^{2l} \text{ nem fa}] \cdot p(N_{e_1}^{2l} \text{ nem fa}). \tag{5.16}$$

A 2. Lemmából tudjuk, hogy $p(N_{e_1}^{2l} \text{ nem fa}) \leq \frac{\gamma}{n}$. Továbbá $E[Z_1|N_{e_1}^{2l} \text{ fa}] = p$. Ezeket és triviális 1 és 0 felsőbecsléseket használva adódik az (5.15)-ből:

$$nd_v p\left(1 - \frac{\gamma}{n}\right) \leq E[Z] \leq nd_v p\left(1 + \frac{\gamma}{n}\right). \tag{5.17}$$

Ebből pedig átrendezéssel adódik $n > \frac{\gamma}{n}$ esetén az (5.13).

(a): Be fogunk vezetni az Ω elemein P_i, $0 \leq i \leq d_v n + n := s$, ekvivalenciarelációk finomodó sorozatát. Vagyis teljesülni fog, hogy ha $(\pi_1, r_1), (\pi_2, r_2) \in \Omega$ közös ekvivalenciaosztályban vannak P_i szerint, akkor minden $j < i$-re P_j szerint is közös ekvivalenciaosztályban vannak. P_o szerint legyen minden elem ekvivalens egymással. P_i szerint, ha $i \leq nd_v$ legyenek $(\pi_1, r_1), (\pi_2, r_2)$ ekvivalensek, ha π_1 és π_2 permutációkra $k \leq i$-re $\pi_1(k) = \pi_2(k)$, vagyis ha az első i él ugyanaz az elemi eseménynek megfelelő Tanner-Gráfban. Ha $nd_v < i$, akkor $(\pi_1, r_1), (\pi_2, r_2)$ ekvivalensek, ha a permutációk megegyeznek, továbbá az r_1 és az r_2, vagyis a csatorna kimenetek, első $i - nd_v$ koordinátája megegyezik. Legyen Σ_i az i. ekvivalencia által generált σ algebra, vagyis az a legszűkebb σ algebra, amely tartalmazza a P_i ekvivalenciareláció minden ekvivalenciaosztályát. Ezekkel legyen Z_i a Z valószínűségi változó Σ_i σ algebrára való feltételes várható értéke, $Z_i := E[Z|\Sigma_i]$. Ekkor $Z_0, \ldots, Z_s$ Doob Martingál sorozat. Vegyük észre, hogy $Z_0 = E[Z]$, $Z_s = Z$.

Ha belátjuk, hogy $|Z_{i+1} - Z_i| \leq 8(d_v d_v)^l$ ha $0 \leq i < nd_v$, $nd_v \leq i$-re pedig azt, hogy $|Z_{i+1} - Z_i| \leq 2(d_v d_v)^l$, akkor a 8. Tételben ismertetett Azuma-egyenlőtlenségből következni fog az állítás (a) pontja.

Vizsgálódjunk először $0 \leq i < nd_v$ esetén. A tankönyvi definíciónál kicsit közelebb szeretném az olvasót hozni a feltételes várható érték fogalmához. Ha olyan σ algebrára nézve nézzük valószínűségi változók várható értékét, amelyet egy olyan Ω partíció generál, amelybe tartozó halmazok pozitív mértékűek, akkor a feltételes várható érték egy olyan valószínűségi változó, amely a generáló partíciókon konstans, a konstans értéke pedig az eredeti valószínűségi változó generáló halmazon vett Lebesgue integrálja elosztva a halmaz Lebesgue mértékével. Más megfogalmazásban a konkrét példánkra visszatérve, legyen (π, r) a P_i ekvivalencia egy tetszőleges H ekvivalenciaosztájának eleme. Ekkor:

$$Z_i(\pi, r) = E[Z|\Sigma_i](\pi, r) = E[Z|H]. \tag{5.18}$$

Tegyük fel, hogy a P_{i+1}. partícióban a H ekvivalenciaosztály $K_1, \ldots, K_t$ ekvivalenciaosztályokra bomlik. Ekkor $(\pi, r) \in H$ esetén az (5.18)-ből következik:

$$E[Z|\Sigma_i](\pi, r) = E[Z|H] = \sum_{j=1}^{t} E[Z|K_j]p(K_j|H) = \sum_{j=1}^{t} Z_{i+1}(K_j)p(K_j|H). \tag{5.19}$$

Ez az utóbbi egyenlőség közel hozott minket Z_{i+1} és Z_i különbségének becsléséhez.

Definiálok egy mértéktartó bijekciót a fenti halmazok közül két tetszőleges K_x és K_y halmaz között. A K_x és K_y-ban olyan elemi események vannak, amelyekhez tartozó permutációk első i koordináta ugyanaz, továbbá tekintve, hogy különböző P_{i+1}-beli ekvivalenciaosztályok, ezért az $i+1$. koordináta szükségképpen különbözik. Tegyük fel, hogy a K_x-beli elemi eseményeknél az $i+1$. koordináta x, a K_y elemi eseményeinél pedig y-n. Ekkor egy $(\pi, r) \in K_x$ elemi eseményhez hozzárendeljük a $(\pi^*, r) \in K_y$ elemi eseményt, ahol π^*-t a következő módon képezzük. Megkeressük azt az r koordinátát, amire $\pi(r) = y$. Tekintve, hogy K_x és K_y-hoz tartozó permutációk első i koordinátája megegyezik, $i+1 < r$. Ezek alapján π^* legyen olyan, hogy $\pi^*(i+1) = y$, és $\pi^*(r) = x$, a többi koordináta egyezzen meg π koordinátáival. Ez a megfeleltetés nyilvánvalóan mértéktartó. Továbbá megvan a következő nagyon jó tulajdonsága:

$$|Z(\pi, r) - Z(\pi^*, r)| \leq 8(d_v d_c)^l. \tag{5.20}$$

Utóbbi egyenlőtlenséget úgy láthatjuk be könnyen, hogy felidézzük, hogy a két elemi eseménynek megfelelő Tanner-Gráf összesen négy élben különbözik, a leképezés során két élet töröltünk, két új élet felvettünk. A dekódoló bemenetei mindkét esetben ugyanazok. Így csak azok az l. iterációs (v, c) üzenetek különböznek a két gráfban, amelyek $N^{2l}_{(v,c)}$ környezetében szüntettünk meg egy élet, vagy amelyek környezetéhez hozzávettünk egy élet. Egyszerű kombinatorikai megfontolásokból következik, hogy egy él legfeljebb $2(d_v d_c)^l$ élnek lehet benne az irányított $2l$ mélységű környezetében. Így adódik az (5.20). Abból pedig következik a mértéktartást is figyelembe véve, hogy $Z_{i+1}(K_x)$ és $Z_{i+1}(K_y)$ is csak legfeljebb $8(d_v d_c)^l$-vel térhet el egymástól. Így az (5.19)-ből látjuk, hogy $Z_i(H)$ egymástól kölcsönösen legfeljebb $8(d_v d_c)^l$ távolságra levő számok átlaga, így adódik $|Z_{i+1} - Z_i| \leq 8(d_v d_v)^l$. Kész vagyunk a $0 \leq i < nd_v$ esettel. Az $nd_v \leq i$ esetben hasonlóan lehet eljárni, ott azt kell megvizsgálni, hogy egy csatornabemenet megváltozása mennyi élet befolyásol. ■

Röviden összefoglalom a részt, egyben kifejtem a bebizonyított koncentrációs tétel jelentőségét. Rögzített d_v, d_c mellett vizsgáltuk a $C^n(d_v, d_c)$-ből véletlenül választott kódot. Szimmetria feltételek miatt elég volt csak az azonosan 1 csatorna bemenettel foglalkoznunk. Beláttuk, hogy körmentességet feltételezve lehetséges az éleken haladó üzenetek eloszlását numerikusan számolni. A korlátos fokszám miatt ezek az eloszlások nagyon szoros kapcsolatban vannak a lokális körmentesség melleti betűnkénti hibavalószínűséggel. Említettem, hogy paraméterezett csatornacsaládnál megfelelő feltételekkel van egy határparaméter, ami alatt körmentességet feltétezve a betűnkénti hibavalószínűség 0-hoz tart. A fenti koncentrációs tételből pedig, ha a (c) részben a valószínűségen belül osztunk nd_v-vel, levonhatjuk azt a következtetést, hogy tetszőleges a határparaméternél kisebb paraméterű csatornára, rögzített ε-hoz találunk olyan $l(\varepsilon)$ iterációs számot, és olyan $N(\varepsilon, l)$-t, amelyre teljesül, hogy $N < n$-re, a $C^n(d_v, d_c)$-ből véletlenül választott kód n-ben exponenciálisan növekvő valószínűséggel ε-nál kisebb betűnkénti hibavalószínűséggel dekódol l iterációs lépést elvégezve a Belief Propagation dekódolóval. Tekintve, hogy a csatornákhoz tartozó határparaméter nagyon közel van a Shannon-tétel által biztosított elméleti határhoz, azt a következtetést vonhatjuk le, hogy az LDPC kód kiegészítve Belief Propagation alapú dekódolóval a Shannon-kapacitáshoz nagyon közel teljesít.

Megjegyzem, hogy [21]-ben irreguláris LDPC kódokkal sikerült a Turbó-kódoknál is jobban megközelíteni a kapacitást. Irreguláris LDPC kód annyiban különbözik a reguláristól, hogy ott a d_v és a d_c konstansok csak felsőkorlátjai a fokszámnak, továbbá a gráfban szereplő fokszámok eloszlása adott. Urbanke megjegyzi, hogy minden [20]-ban ismertetett eredmény minden további nélkül igaz marad irreguláris LDPC kódokra is. A koncentrációs tételben igazán csak a fokszám korlátossága játszott fontos szerepet. Viszont a körmentesség mellett az üzenetek nemcsak a csatorna miatt véletlenek, hanem a kód randomizálása miatt is, hiszen konkrét irreguláris kódok lokálisan eltérhetnek, csak valószínűségi értelemben hasonlóak.

Teszek egy érdekes megjegyzést. A részben levezetett tételek nem mondanak semmit a Belief Propagation konvergenciájával kapcsolatban. Elvben elképzelhető, hogy az LDPC kód úgy dekódol jól, hogy a dekódolón haladó üzenet oszcillál a jó döntési tartományban.

Végezetül megjegyzem, hogy az itt használt alapelveket kiterjesztették Turbó-kódra is. Így elmondható, hogy nagy blokkméret esetén, ha a véletlen permutáció előállítása problémamentes, van matematikai háttér a két empirikusan jónak bizonyuló kód mögött.

6. fejezet

Összefoglalás

A dolgozatban a sok formában megjelenő Belief Propagation algoritmusról adtam rendezett, egységes képet. Bemutattam a hozzá kapcsolódó alapvető fogalmakat, alkalmazásokat és nyitott kérdéseket. A kódolási alkalmazásokra különös figyelmet fordítottam. Tudomásom szerint nincs magyar nyelvű szakirodalma a témának, így hiánypótló a dolgozat. A témakör ismertetése közben tettem saját megjegyzéseket, kiegészítéseket. Igyekeztem hangsúlyozni, hogy az együttes eloszlásban szereplő 0-k fontosak a gyakorlati alkalmazásokban, például változók megfigyelése szükségképpen együtt jár 0-k megjelenésével. Továbbá rámutattam, hogy az algoritmus különböző megjelenési formáinak ekvivalenciájában fontos az ütemezés szerepe. A 2.3 részben szereplő Markov-Faktor-Gráf definíció a saját ötletem, segíti a különböző megjelenési formák egységesebb leírását. A 4.1 részben az ütemezéseket három csoportba osztottam, továbbá rámutattam néhány alapvető jelenségre. A 4.2 részben abban a speciális esetben, amikor a szóban forgó grafikus reprezentáció csak egy kört tartalmazott, az ütemezés és a változók megfigyelt értékei tekintetében kicsit általánosabban bizonyítottam a szakirodalomban már létező tételt. A 4.3 részben különös figyelmet fordítottam arra, hogy az ott bizonyított 5. Tétel kis módosításával kijöjjön a 7. Állítás.

A jövőben ha lesz rá lehetőségem részletesen fogok foglalkozni a 0-k és az ütemezés szerepével. [30] foglalkozik a 4.3 részben tárgyalt kigombolyított fa általánosításával párhuzamostól eltérő ütemezés mellett. A tárgyalást nem érzem teljesnek. Sok lehetőséget látok ebben az irányban. Ezenkívül fontos feladat már létező konvergenciát biztosító eredmények kapcsolatának pontos feltárása, ezt követően pedig általánosabb összefüggések keresése különös tekintettel a becslés és az igazi marginális függvények viszonyára.

Köszönetnyilvánítás

Szeretném megköszönni témavezetőmnek, Dr. Csiszár Imrének, a dolgozat készítése közben nyújtott sok segítségét.

Irodalomjegyzék

[1] R.G.Gallager, *Low-Density Parity-Check Codes*, Cambridge, MA: MIT Press, 1963.

[2] A.J.Viterbi, *Error Bounds for Convolutional Codes and an Asymtotically Optimum Decoding Alghorithm*, IEEE Trans. Inform. Theory, vol. IT-13, pp. 260-269, Apr. 1967.

[3] L.R.Bahl, J.Cocke, F.Jelinek, J.Raviv, *Optimal Decoding of Linear Codes for Minimizing Symbol Error Rate*, IEEE Trans. Information Theory, vol. IT-20, pp. 284-287, Mar. 1974.

[4] R.J.McEliece, *The Theory of Information and Coding, Chapter 9. Convolutional Codes*, Addison-Wesley Publishing Company, 1977.

[5] R.M.Tanner, *A Recursive Approach to Low Complexity Codes*, IEEE Trans. Inform. Theory, vol. IT-27, pp. 533-547, Sept. 1981.

[6] J.Pearl, *Probabilistic Reasoning in Intelligent Systems*, 2nd ed. San Francisco, CA: kaufmann, 1988.

[7] C.Berrou, A.Glavieux, P.Thiimajshima, *Near Shannonlimit Error-Correcting Coding and Decoding: Turbo Codes*, in proc. 1993 IEEE Int. Conf. Communications, Geneva, Switzerland, pp. 1064-1070, May 1993.

[8] Stoyan Gisbert, *Numerikus módszerek*, ELTE Typotex, 1993.

[9] D.J.C.Mackay, R.M.Neal, *Good Codes Based on Very Sparse Matrices*, in Proc. 5th IMA Conf. Cryptography and Coding, C. Boyd, Ed. Berlin, Germany: Springer Lecture Notes in Computer Science, vol. 1025, pp. 100-111, 1995.

[10] N.Wiberg, *Codes and Decoding on General Graphs*, Ph.D. dissertation, Linköping Univ., Linköping, Sweden, 1996.

[11] S.M.Aji and R.McEliece, *A General Algorithm for Distributing Information on a Graph*, in Proc. 1997 IEEE int. Symp. Information Theory, Ulm, Germany, p. 6., July 1997.

[12] R.J.McEliece, D.J.C.Mackay, Jung-Fu Cheng, *Turbo Decoding as an Istance of Pearl's "Belief Propagation" Algorithm*, IEEE Journal On Selected Areas In Communications, Vol.16, no.2, Feb. 1998.

[13] S.M.Aji, G.B.Horn, R.J.McEliece, *On the Convergence of Iterative Decoding on Graphs with a Single Cycle*, in Proc. 1998 IEEE Int. Symp. Information Theory, Cambridge, MA, p. 276, Aug. 1998.

[14] K.P.Murphy, Y.Weiss, M.I.Jordan, *Loopy Belief Propagation for Approximate Inference: An Empirical Study*, in Proc. Uncertainty in Artificial Intelligence, 1999.

[15] Charan Langton, *Tutorial 12 Coding and Decoding with Convolutional Codes*, http://complextoreal.com/chapters/convo.pdf, July 1999.

[16] S.M.Aji and R.McEliece, *The Generalized Distributive Law*, IEEE Trans. Information Theory, vol. 46, pp. 325-343, Mar 2000.

[17] Y.Weiss, *Correctness of Local Probability Propagation in Graphical Models with Loops*, Neur. Comput., vol. 12, pp. 1-41, 2000.

[18] F.R.Kschischang, B.J.Frey, H.A. Loeliger, *Factor Graphs and the Sum-Product Algorithm*, IEEE Trans. Information Theory, vol. 47, NO. 2, pp. 498-519, February 2001.

[19] G.D.Forney, Jr., *Codes on Graphs: Normal Realizations*, IEEE Trans. Information Theory, vol. 47, NO. 2, pp. 520-548, February 2001.

[20] T.J.Richardson, R.L.Urbanke, *The Capacity of Low-Density Parity-Check Codes Under Message-Passing Decoding*, IEEE Trans. Information Theory, vol. 47, pp. 599-618, Feb. 2001.

[21] T.J.Richardson, M.A.Shokrollahi, R.L.Urbanke, *Design of Capacity-Approaching Irregular Low-Density Parity-Check Codes*, IEEE Trans. Information Theory, vol. 47, pp. 619-637, Feb. 2001.

[22] T.J.Richardson, R.L.Urbanke, *Efficient Encoding of Low-Density Parity-Check Codes*, IEEE Trans. Information Theory, vol. 47, pp. 638-656, Feb. 2001.

[23] Y.Weiss, W.T.Freeman *On the Optimality of Solutions of the Max-Product Belief-Propagation Algorithm in Arbitrary Graphs*, IEEE Trans. Information Theory, vol. 47, pp. 736-744, Feb. 2001.

[24] S.C.Tatikonda, M.I.Jordan, *Loopy Belief Propagation and the Gibs Measure*, in Proc. 18th Annu. Conf. Uncertainty in Artificial Intelligence, San Francisco, CA, pp. 493-500, 2002.

[25] F.R.Kschischang, *Codes Defined on Graphs*, IEEE Communications Magazine, August 2003

[26] T.Heskes, *On the Uniqueness of Loopy Belief Propagation Fixed Points*, Neur. Comput., vol. 16, pp. 2379-2413, Nov. 2004.

[27] A.T.Ihler, J.W.Fisher, A.S.Willsky, *Loopy Belief Propagation: Convergence and Effects of Message Errors*, J. Machine Learning Res., vol. 6., pp. 905-936, 2005.

[28] J.S.Yedidia, W.T.Freeman, Y.Weiss, *Constucting Free-Energy Approximations and Generalized Belief Propagation Algorithms*, IEEE Trans. Inf. Theory, vol.51, no.7, pp.2282-2312, Jul. 2005.

[29] Christofer M. Bishop, *Pattern recognization and machine learning, Chapter 8. Graphical Models*, Springer, 2006.

[30] N.Taga, S.Mase, *On the convergence of loopy belief propagation alghorithm for different update rules*, IEICE Trans. Fundamentals, vol. E89-A, no. 2, pp. 575-582, Feb. 2006.

[31] G.Elidan, I.McGraw, D.Koller, *Residual Belief Propagation. Informed Scheduling for Asynchronous Message Passing*, in Proc. 22nd Annu. Conf. Uncertainty in Artificial Intelligence (UAI-06), Boston, MA, Jul. 2006.

[32] D.J.Costello, G.D.Forney, *Channel Coding: The Road to Channel Capacity*, Proceedings of the IEEE, Vol. 95, No. 6, June 2007.

[33] J.M.Mooij, H.J.Kappen, *Sufficient Conditions for Convergence of the Sum-Product Algorithm*, IEEE Trans. Information Theory, vol. 53, pp. 4422-4437, Dec. 2007.

[34] Csiszár Imre BME-n tartott információelmélet előadása (BMETE955201), 2007/2008-as tanév első félév.

[35] Ivanyos Gábor BME-n tartott algebrai kódelmélet (BMETE915008) előadása, 2007/2008-as tanév első félév.

[36] J.M.Walsh, P.A.Regalia, *Belief Propagation, Dykstra's Algorithm, and Iterated Information Projections*, submitted to IEEE Trans. Inform. Theory, June 10, 2008.