Mesterséges Intelligencia I. gyakorlat

Dobó András

2013/2014 I. félév

Felhasznált irodalom:

- Az előadás jegyzete (http://www.inf.u-szeged.hu/~jelasity/mi1/2013/jegyzet.pdf)
- Peter Norvig, Stuart J. Russel: Mesterséges intelligencia - Modern megközelítésben
- Példa feladatok egy része korábbi gyakorlati anyagból illetve az internetről

1. óra - Követelmények és tematika

A kurzus teljesítésének feltételei

A kurzus teljesítésének értékelése pontozás alapján történik. A gyakorlat és előadás értékelése külön jeggyel történik. A maximálisan összegyűjthető pontszám a gyakorlaton 50. Az előadás teljesítésének a feltétele a sikeres gyakorlat. A gyakorlat kreditértéke az eredetileg 4 kredites kurzus esetén 1, eredetileg 6 kredites kurzus esetén 2 és eredetileg 7 kredites kurzus esetén 3.

A gyakorlat végső értékelése (gyakorlati jegy) a következő:

0-22 pont:	elégtelen (1)
23-30 pont:	elégséges (2)
31-36 pont:	közepes (3)
37-42 pont:	jó (4)
43- pont:	jeles (5)

Gyakorlati pontszám

Két röpdolgozat (3+3 pont) és zárthelyi dolgozat (20 pont). A röpdolgozatok időpontja véletlenszerű. A zárthelyi dolgozat során minimálisan összesen 10 pont elérése kötelező. Javításra illetve pótlásra csak egy alkalommal van lehetőség. Igazolt hiányzás esetén a hiányzás alatt megírt dolgozat(ok) pótolható(ak). Javítani csak akkor lehet ha nincs meg a 10 pont. A legalább 30%-ra megírt javító dolgozat minősül sikeresnek, de ekkor is az eredeti pontszám marad érvényes, függetlenül a javító dolgozatban elért pontszámtól.

Később ismertetésre kerülő kötelező program megírása (24 pont). A később pontosítandó végső leadási határidő november végére várható. A programok teljesítményeik alapján kerülnek pontozásra. Legalább 12 pont elérése kötelező. Aki a megadott határidőkre nem készíti el a programot, illetve amennyiben a program a minimális követelményeknek nem tesz eleget, a kötelező program teljesítése sikertelen. A programoknak önálló munkáknak kell lenni. A kötelező program beadása nem halasztható, utólagos pótlása, javítása nem megengedett.

A kötelező programból szúrópróbaszerűen kiválasztott hallgatók szóban is kötelesek beszámolni, a kiválasztott hallgatókat erről a viszgaidőszak kezdetén értesítjük.

A gyakorlatok látogatása kötelező. A hiányzás igazolható, amennyiben a hallgató a hiányzást követő gyakorlaton az igazolásra vonatkozó dokumentumot bemutatja az oktatónak. Az igazolt hiányzások száma nem lehet 3-nál több.

Tematika

1. gyakorlat	Követelmények és tematika
2. gyakorlat	Kereséssel történő problémamegoldás elemei és állapottér reprezentáció
3. gyakorlat	Informálatlan keresés
4. gyakorlat	Informált keresés
5. gyakorlat	Játékok
6. gyakorlat	Felügyelt tanulás
7. gyakorlat	Naiv-Bayes osztályozás
8. gyakorlat	Számítógépes nyelvfeldolgozás, szövegek automatikus osztályozása
9. gyakorlat	Döntési fák
10. gyakorlat	Lokális keresés, optimalizálás
11. gyakorlat	Bayes-hálók
12. gyakorlat	Gyakorlás

2. óra - Kereséssel történő problémamegoldás elemei és állapottér reprezentáció

Ágensek, feladatkörnyezet és problémák modellezése:

Ágens: valami, ami cselekszik

Racionális ágens: egy olyan ágens, amely a tudásához viszonyítva helyesen cselekszik

Problémamegoldó ágens: olyan célorientált ágens, mely úgy határozza meg, mit is kell tennie, hogy olyan cselekvéssorozatokat keres, amelyek a kívánt célállapotokba vezetnek

A vizsgált problémáknál feltesszük, hogy a **feladatkörnyezet**:

- **diszkrét**: állapotok, időkezelés stb. nem folytonosak
- **statikus**: a környezet változatlan amíg az ágens gondolkodik
- **teljesen megfigyelhető**: az ágens a szenzorjai segítségével a környezet teljes állapotát ismeri
- **determinisztikus**: a környezet következő állapotát a jelenlegi állapota és az ágens által végrehajtott cselekvés teljesen meghatározza

Ilyen esetben a **problémák tökéletesen modellezhetők** a következőkkel:

- **lehetséges állapotok halmaza**
- **kezdőállapot**: az ágens ebből kezdi a cselekvését
- **lehetséges cselekvések halmaza**: legáltalánosabb leírása az állapotátmenet függvény egy x állapothoz hozzárendel egy <cselekvés, utódállapot> párok halmazát
- **állapotátmenet költségfüggvénye**: <állapot, cselekvés, utódállapot> hármashoz hozzárendel egy valós számot, a költséget
- **célállapotok halmaza**: az állapottér részhalmaza

A fenti modell egy irányított, súlyozott gráfot definiál, ahol a csúcsok az állapotok, az élek cselekvések, a súlyok pedig a költségek. Ez a gráf az **állapottér**.

Minden problémát végtelen különböző módon modellezhetünk. A probléma megoldásának bonyolultsága függ az alkalmazott modelltől. A gyakorlatban fontos, hogy minél hatékonyabban, minél kevesebb állapottal modellezzük a problémákat, mert ez nagyban csökkenti megoldásuk bonyolultságát.

Példa problémák modellezése

8 királynő probléma (1. megoldás)

Cél: 8 királynő elhelyezése a sakktáblán úgy, hogy azok ne támadják egymást.

A probléma egy lehetséges modellje:

- lehetséges állapotok halmaza: n ($0 \leq n \leq 8$) királynő tetszőleges elrendezése a táblán
- kezdőállapot: a táblán nincs egyetlen királynő sem
- lehetséges cselekvések halmaza: egy királynő elhelyezése egy üres mezőre
- állapotátmenet költségfüggvénye: lényegtelen, csak a célállapot számít
- célállapotok halmaza: olyan állapotok halmaza, ahol a királynők nem támadják egymást

8 királynő probléma (2. megoldás)

Cél: 8 királynő elhelyezése a sakktáblán úgy, hogy azok ne támadják egymást.

A probléma egy lehetséges modellje:

- lehetséges állapotok halmaza: n ($0 \leq n \leq 8$) királynő olyan elrendezése a táblán, hogy az n bal oldali oszlopban oszloponként egy található úgy, hogy nem támadják egymást
- kezdőállapot: a táblán nincs egyetlen királynő sem
- lehetséges cselekvések halmaza: egy királynő elhelyezése a tábla bal széléhez a legközelebbi, még üres oszlopba úgy, hogy azt ne támadja egyetlen királynő se
- állapotátmenet költségfüggvénye: lényegtelen, csak a célállapot számít
- célállapotok halmaza: olyan állapotok halmaza, ahol a királynők nem támadják egymást

Ez a modell a probléma állapotterét az előző modellel szemben 1,8 x 10^{14}-ről 2057 méretű térré csökkenti. Ez a probléma megoldásának bonyolultságát is hasonló mértékben csökkenti. Ezért van szükség arra, hogy a problémákat minél hatékonyabban, minél kevesebb állapottal modellezzük.

Palacsinták sorba rakása

Van n különböző méretű palacsintánk véletlenszerű sorrendben egymás tetejére rakva. Palacsintasütővel bármelyik kettő közé be tudunk nyúlni, és a palacsintasütő feletti részt megfordítani. Cél: a palacsinták növekvő sorrendbe rakása a lehető legkevesebb fordítással.

A probléma egy lehetséges modellje:

- lehetséges állapotok halmaza: n elemű, valós számokból álló permutációk, a számok a palacsinták méretei alulról felfelé sorrendben
- kezdőállapot: egy előre specifikált ilyen permutáció

- lehetséges cselekvések halmaza: egy permutáció hátsó p elemének fordított sorrendbe rendezése
- állapotátmenet költségfüggvénye: egy forgatás költsége 1
- célállapotok halmaza: egy olyan permutáció, ahol a számok csökkenő sorrendben vannak

Utazástervezési probléma

Egy valós életbeli probléma egyszerűsített változata (általában a valós problémákat nehéz pontosan specifikálni). Egy adott repülőtérről adott időpontban minél gyorsabban és olcsóbban akarunk eljutni egy másik repülőtérre repülővel. A gyakorlatban ez sokkal bonyolultabb, egyéb tényezőktől is függ.

A probléma egy lehetséges modellje:

- lehetséges állapotok halmaza: a világban az összes lehetséges repülőtérből és összes lehetséges időpontból álló párok halmaza
- kezdőállapot: egy előre specifikált repülőtér és idő pár
- lehetséges cselekvések halmaza: az adott repülőtérről az adott időnél később induló repülőgépjáratok alkalmazása
- állapotátmenet költségfüggvénye: a repülőúttal (és várakozással) eltelt idő és a repülőjegy költségének függvénye
- célállapotok halmaza: olyan repülőtér és idő párok halmaza, ahol a repülőtér rögzített az időnek pedig van egy maximális korlátja

8-as kirakójáték

A feladat a 8-as kirakójáték legkevesebb mozgatással való kirakása. A játék célja egy 3x3-as táblán kezdetben véletlenszerűen elhelyezett 8 darab, 1-től 8-ig számozott kocka olyan helyzetbe mozgatása, hogy az üres hely a bal felső sarokban legyen, a számozott kockák pedig sorrendben következzenek. Egy mozgatás során egy kockát lehet az oldallapjával szomszédos üres mezőre mozgatni.

A probléma egy lehetséges modellje:

- lehetséges állapotok halmaza: a nyolc kocka és az üres hely pozíciója
- kezdőállapot: egy véletlen állapot
- lehetséges cselekvések halmaza: az üres helynek a 4 irány közül valamely irányba történő legális mozgatása (nyilván a tábla széléről nem mozgathatjuk le)
- állapotátmenet költségfüggvénye: egy mozgatás költsége 1
- célállapotok halmaza: egy olyan állapot, ahol az üres hely a bal felső sarokban van, a számokat jelülő kockák pedig sorrendben következnek

További problémák:

Az előadásjegyzetben, a tankönyvben és a segédanyagok közt találhatók.

3. óra - Informálatlan keresés

Megoldások keresése

A problémák modellje, azon belül is a lehetséges állapotok halmaza, a lehetséges cselekvések költsége és az állapotátmenet költségfüggvénye, egy súlyozott gráfot definiál, ahol a csúcsok az állapotok, az élek cselekvések, a súlyok pedig a költségek. Ez a gráf az **állapottér**.

Az ágensek feladata az, hogy adott kezdőállapotból találjon egy **minimális költségű utat** egy célállapotba. Ez az állapottérben végrehajtott kereséssel történik. Nem egészen a klasszikus legrövidebb út keresési probléma: az állapottér nem mindig adott explicit módon, és végtelen is lehet.

Az **informálatlan keresés** azt jelenti, hogy ezen stratégiáknak semmilyen információjuk nincs az állapotokról a probléma definíciójában megadott információn kívül.

Keresőfa

Ötlet: **keresőfa**, azaz a kezdőállapotból növesszünk egy fát a szomszédos állapotok hozzá vételével, amíg célállapotot nem találunk.

Vigyázat: a keresőfa nem azonos a feladat állapotterével! Pl. az állapottér nem is biztosan fa, amely esetben a keresőfa nőhet végtelenre is, akkor is, ha az állapottér véges.

Csomópontok reprezentációja:

- **Állapot**: az állapottérnek a csomóponthoz tartozó állapota
- **Szülő-csomópont**: a keresési fa azon csomópontja, amely a kérdéses csomópontot generálta
- **Cselekvés**: a csomópont szülő-csomópontjára alkalmazott cselekvés
- **Út-költség**: a kezdeti állapotból a kérdéses csomópontig vezető út általában g(n)-nel jelölt költsége, ahogy ezt a szülőmutatók jelzik
- **Mélység**: a kezdeti állapotból vezető út lépéseinek a száma.

Algoritmus:

```
fakeresés
1 perem <- {új-csúcs(kezdőállapot)}
2 if perem.üres() return failure
3 csúcs <- perem.elsőkivesz()
4 if csúcs.célállapot() return csúcs
5 else perem.beszúr(csúcs.kiterjeszt())
6 goto 2
```

A `csúcs.kiterjeszt()` generálja az aktuális állapotból elérhető állapotok halmazát.

A `perem` (más néven `nyílt halmaz`) egy prioritási sor, ami a már legenerált, de még kifejtésre váró csomópontokat tárolja. Ez (pontosabban az, hogy a peremből milyen sorrendben vesszük ki a csúcspontokat, vagyis a `perem`-nek az `elsőkivesz()` függvénye) definiálja a keresési stratégiát.

Algoritmusok hatékonyságának vizsgálata

Egy algoritmus **teljes** akkor és csak akkor, ha minden esetben, amikor létezik véges számú állapot érintésével elérhető célállapot, az algoritmus meg is talál egyet.

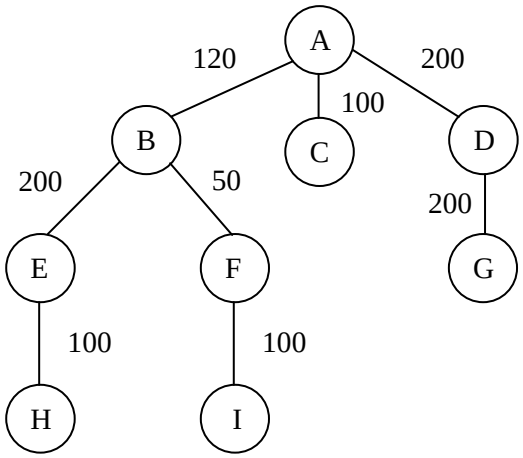
Egy algoritmus **optimális** akkor és csak akkor, ha teljes, és minden megtalált célállapot optimális költségű.

Az **idő- és memóriaigényt** nem az állapottér méretének függvényében vizsgáljuk, hanem a speciális alkalmazásunkra (MI) szabva a következő paraméterek függvényében: b (elágazási tényező): szomszédok maximális száma, m: keresőfa maximális mélysége, d: a legkisebb mélységű célállapot mélysége a keresőfában. Feltesszük hogy b véges, m és d lehet megszámlálhatóan végtelen!

Fa-kereső algoritmusok

Az algoritmusok csak a perem megvalósításában, és így keresési stratégiájukban különböznek.

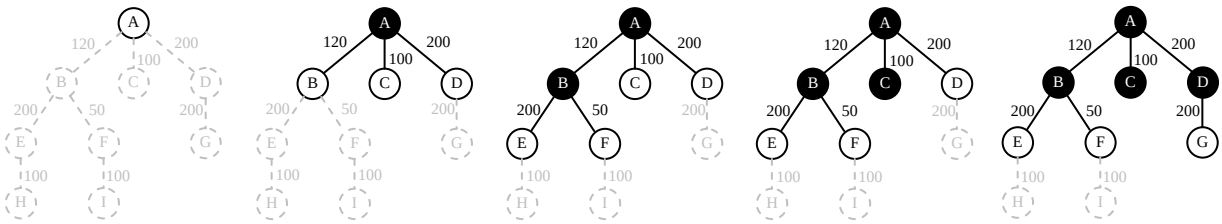
Algoritmusok példa problémán való megoldása. Probléma állapottere:

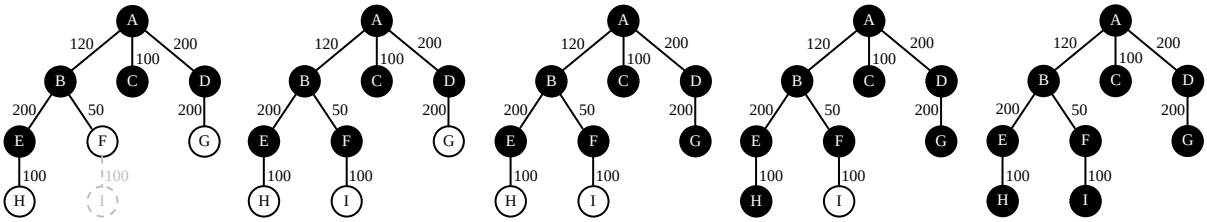


Cél: A-ból I-be legkisebb költségű út megtalálása

Szélességi keresés

A perem: rendezése FIFO (first-in-first-out), vagyis a legelőször bekerült csomópont kerül ki a peremből legelőször





Az ábrákon a szaggatottal jelölt csomópontokat még nem derítette fel az algoritmus, azok még nem kerültek be a perembe, az üres körrel jelölt csomópontok vannak benne a peremben, a teli körrel jelölt csomópontok pedig már kikerültek a peremből.

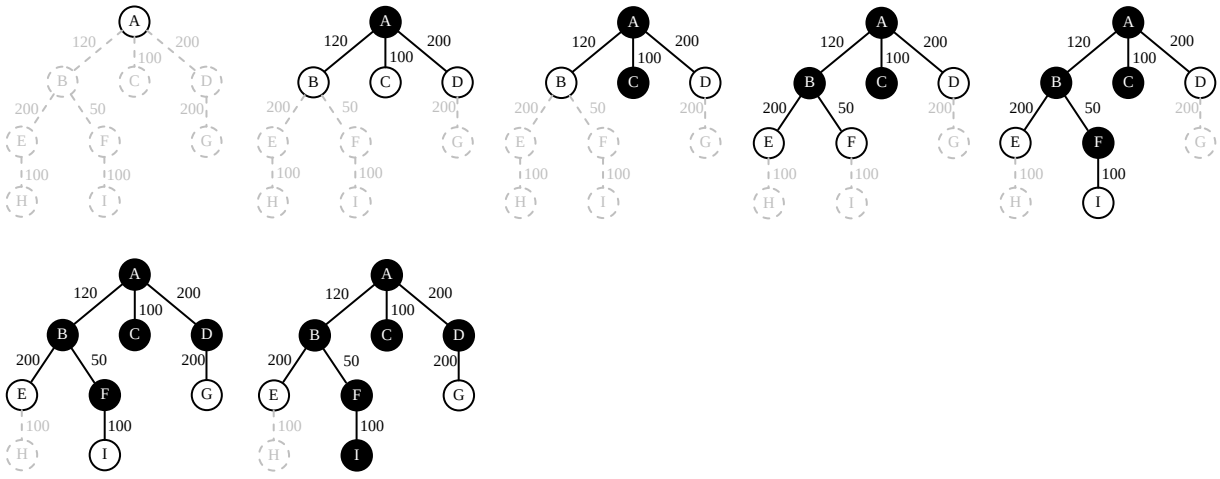
Kifejtett csomópontok sorrendje: A, B, C, D, E, F, G, H, I

Hatékonyság:

- teljes
- csak akkor optimális, ha az útköltség a csomópont mélységének nem csökkenő függvénye
- időigény=tárigény=$O(b^{d+1})$ (gyakorlatban általában gyorsan kifut a memóriából)

Egyenletes költségű keresés

A perem: rendezése költség alapú, először a legkisebb költségű csúcsot vesszük ki



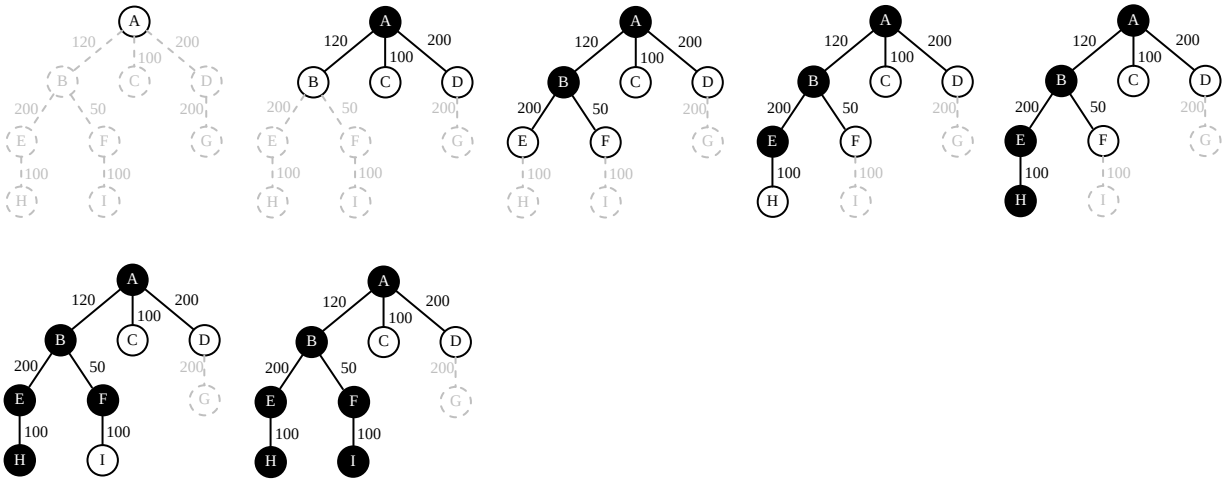
Kifejtett csomópontok sorrendje: A, C, B, F, D, I

Hatékonyság:

- csak akkor teljes és optimális, ha minden él költsége $\geq \varepsilon > 0$
- időigény=tárigény=$O(b^{1+[C^*/\ \varepsilon]})$, ahol C* az optimális megoldás költsége (sokkal több lehet, mint $O(b^d)$)

Mélységi keresés

A perem: rendezése LIFO (last-in-first-out), vagyis a legutoljára bekerült csomópont kerül ki a peremből legelőször



Kifejtett csomópontok sorrendje: A, B, E, H, F, I

Hatékonyság:

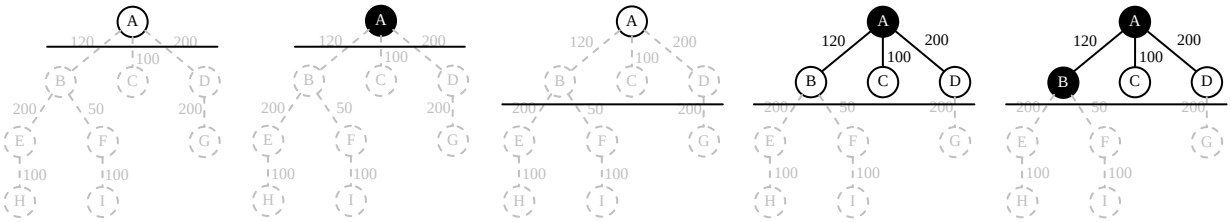
- csak akkor teljes, ha a keresési fa véges mélységű (m véges)
- nem optimális
- időigény= $O(b^m)$ (rosszabb, mint az $O(b^d)$, akár végtelen is lehet)
- tárigény=O(b*m) (nagyon jó)

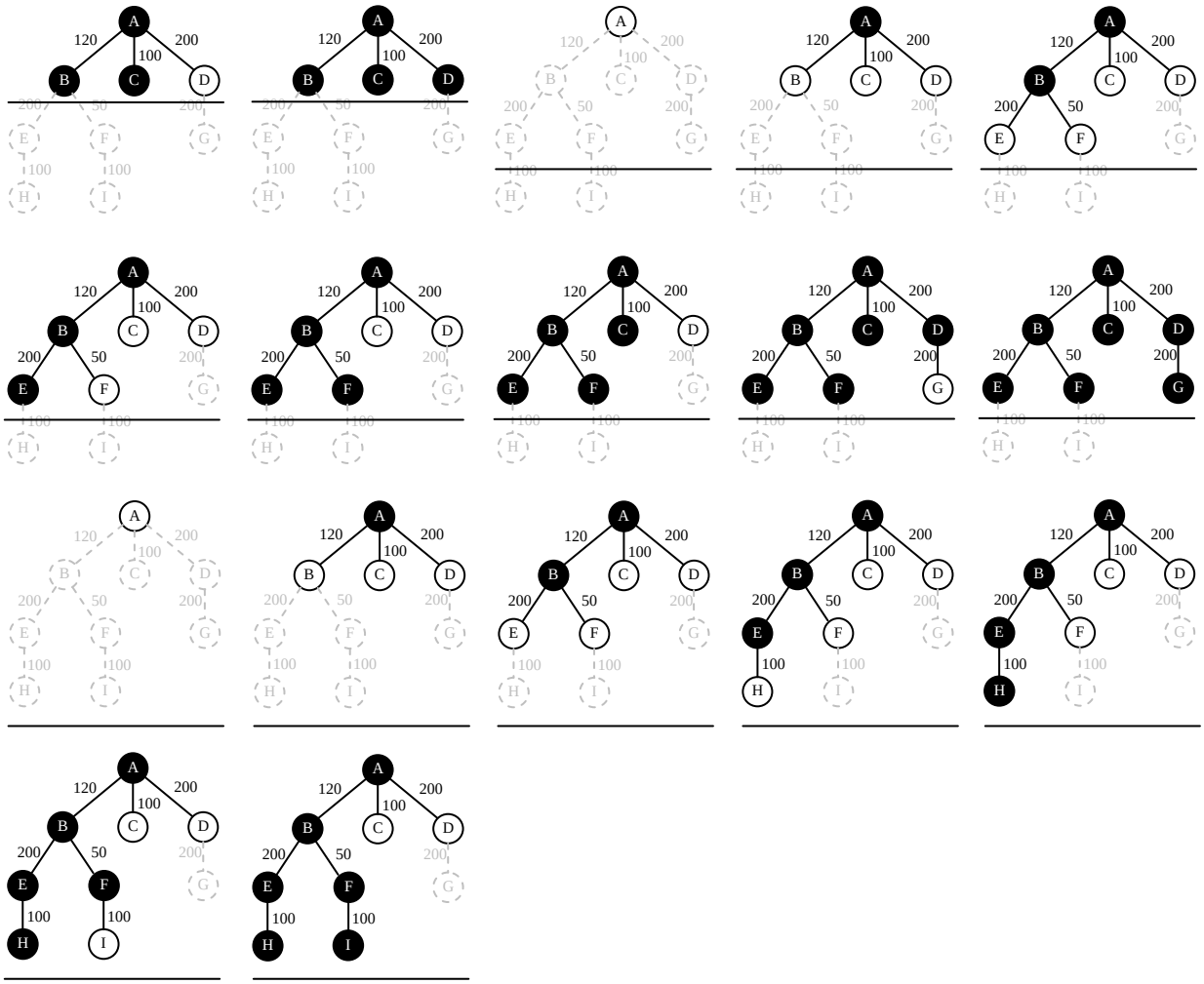
Mélységkorlátolt keresés

A perem: rendezése LIFO, de az utak maximális mélységére van egy korlát, amit rendszerint l-lel jelölünk.

Iteratívan mélyülő keresés

A perem: rendezése LIFO, mélységkorlátolt keresések sorozata egyre növekvő mélységi korláttal





Kifejtett csomópontok sorrendje: A, A, B, C, D, A, B, E, F, C, D, G, A, B, E, H, F, I

Hatékonyság:

- teljesség és optimalitás a szélességi kereséssel egyezik meg
- időigény= $O(b^d)$ (jobb, mint a szélességi)
- tárigény=O(b*d) (jobb, mint a mélységi)
- Ez a legjobb informálatlan kereső (annak ellenére, hogy az első szinteket újra meg újra bejárjuk)

Gráf-kereső algoritmusok

Fa keresés csak akkor hatékony, ha az állapottér maga is egy fa, tehát minden állapotba csak 1 úton lehet eljutni. Ha ez nem teljesül, akkor a fa-kereső algoritmusok hatékonysága az ismétlődő állapotok miatt drasztikusan csökkenhet, végtelen ciklusba is eshetnek.

Megoldás: bármelyik fa-kereső algoritmusnál az ismételt állapotok elkerülhetők egy zárt halmaz segítségével, ami a peremből már egyszer kivett, kiterjesztett csúcsokat tárolja. A perembe ezután csak olyan csúcsot rakunk, ami még nincs benne a zárt halmazban. Ilyen módon a keresési fa véges méretűre vágható.

Algoritmus:

```
gráfkeresés
1 perem <- {új-csúcs(kezdőállapot)}
2 zárt <- {}
3 if perem.üres() return failure
4 csúcs <- perem.elsőkivesz()
5 if csúcs.célállapot() return csúcs
6 else perem.beszúr(csúcs.kiterjeszt() - zárt)
7 zárt.hozzáad(csúcs)
8 goto 3
```

Probléma: mi van, ha egy adott állapothoz a később megtalált út a jobb?
Egyenletes költségű keresésnél bizonyíthatóan optimális a gráf-keresés, ha minden él költsége nemnegatív. Ez ugyanis éppen a Dijkstra algoritmus az állapottérre alkalmazva. (Viszont a teljességhez továbbra is kell, hogy minden költség $\geq \varepsilon > 0$ (nem csak nemnegatív), mert lehetnek végtelen állapotterek.)
Konstans lépésköltségű szélességi keresésnél is optimális.

Kétirányú keresés

Valamely keresési stratégiát alkalmazva egyszerre indít keresést a kiinduló és a célállapotból, akkor áll meg, ha talál közös pontot. Csak akkor alkalmazható, ha a végállapotok halmaza explicit adott.

4. óra - Informált keresés

Informált keresés

Az **informált kereső algoritmusok** a probléma definícióján túlmenően problémaspecifikus tudást is felhasználnak, és ennek segítségével képesek az informálatlan kereső algoritmusoknál hatékonyabban megoldást találni.

Legjobbat-először keresés

Az általános fa-keresés vagy gráf-keresés algoritmusok olyan speciális esete, ahol egy csomópont kifejtésre való kiválasztása az f(n) **kiértékelő függvénytől** függ, általában a legkisebb f(n) értékű csomópont kerül minden lépésben kifejtésre.

Ez a keresés minden esetben a legjobbnak tűnő csomópontot fejti ki. Fontos, hogy a legjobbnak tűnő, és nem pedig a ténylegesen legjobb csomópontot fejti ki, mert ha azt tudnánk, akkor nem is lenne szükségünk keresésre.

Ez egy általános megközelítés, lényegében egy keresési algoritmus család. Több változata létezik, amiket a kiértékelő függvényük különböztet meg egymástól: pl. mohó legjobbat-először keresés, A*.

Az ilyen algoritmusok egy h(n) **heurisztikus függvényt** használnak, amely megbecsli az adott csomóponttól a célig vezető legolcsóbb út költségét. Fontos, hogy ez nem pontos érték, csak becslés, mert ha pontos érték lenne, akkor nem lenne szükségünk keresésre, mindig egyből megtalálnánk a legolcsóbb utat.
Minden problémánál más lesz jó heurisztikus függvény, pl. légvonalbeli távolság a célig a térképen egy útvonal-tervezési problémához jó heurisztika.

Mohó legjobbat-először keresés

A peremben a csomópontok az f(n)=h(n) függvény szerint vannak a rendezve. Tehát az algoritmus azt a csomópontot fejti ki a következő lépésben, amelyiknek az állapotát a legközelebbinek ítéli a célállapothoz.

Ha csak annyit teszünk fel, hogy h(n)=0 ha n célállapot, akkor:

- teljes, de csak akkor, ha a keresési fa véges mélységű (m véges)
- nem optimális
- időigény=tárigény= $O(b^m)$

A legrosszabb eset nagyon rossz, de jó h(n)-nel javítható.

A* algoritmus

A heurisztikus függvényen kívül használ egy g(n)-nel jelölt függvényt is, ami megadja az **aktuális csomópontig megtett út költségét (u.a. mint az útköltség)**.

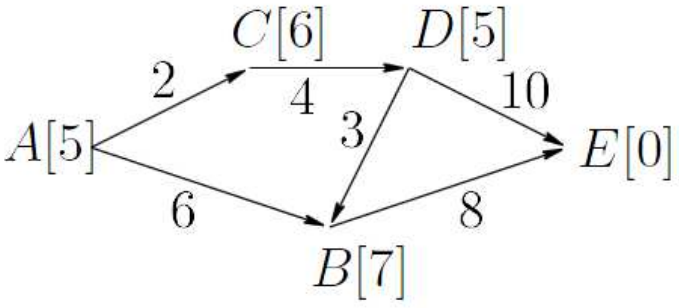
A peremben a csomópontok az f(n)=g(n)+h(n) függvény szerint vannak a rendezve. Tehát az algoritmus azt a csomópontot fejti ki a következő lépésben, amelyiken keresztül vezető legolcsóbb megoldás becsült költsége a legkisebb.

Hatékonyság:

- fa-keresés esetén optimális és teljes, ha a keresési fa véges és h(n) heurisztika **elfogadható**, vagyis ha soha nem becsüli felül a cél eléréséhez szükséges költséget
- gráf-keresés esetén optimális és teljes, ha az állapottér véges és h(n) heurisztika **konzisztens**, vagyis ha $h(n) \leq c(n; a; n') + h(n')$ tetszőleges n-re és annak tetszőleges n' szomszédjára (ez a háromszög egyenlőtlenség egy formája)
- A tárigény általában exponenciális, de nagyon függ a h(n) minőségétől, pl. ha h(n)=h*(n), ahol h*(n) a tényleges hátralevő költség, akkor konstans.
- Az időigény szintén erősen függ a h(n)-től.
- Az A* optimálisan hatékony, tehát egyetlen más optimális algoritmus sem fejt ki garantáltan kevesebb csomópontot, mint az A*.

Megjegyzés: az algoritmusnak van egy olyan változata is, ahol a már bejárt (zárt halmazban levő) csúcspontokat is át lehet linkelni. Ezzel a módosítással az algoritmus akkor is optimális, ha a heurisztika nem konzisztens. Mi az eredeti, az előadáson is használt változattal foglalkozunk.

1. feladat



Kezdőállapot: A, célállapot: E, a szögletes zárójelben levő értékek a heurisztika értékek.

A mohó legjobbat-először keresés és az A* algoritmus gráf-keresési változatát fogjuk alkalmazni. Az alkalmazott heurisztika konzisztens (bizonyítás nélkül), így az A* algoritmus optimális lesz.

Jelölés: n(szülő(n), g(n), f(n))

Megoldás mohó legjobbat-először kereséssel

Nyílt halmaz (perem)	Zárt halmaz
A(NULL, 0, 5)	
B(A, 6, 7), C(A, 2, 6)	A(NULL, 0, 5)
B(A, 6, 7), D(C, 6, 5)	A(NULL, 0, 5), C(A, 2, 6)
B(A, 6, 7), E(D, 16, 0)	A(NULL, 0, 5), C(A, 2, 6), D(C, 6, 5)
B(A, 6, 7)	A(NULL, 0, 5), C(A, 2, 6), D(C, 6, 5), E(D, 16, 0)

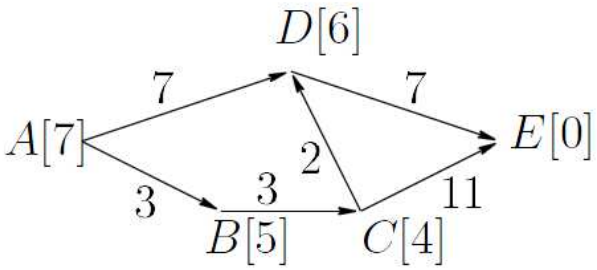
A megtalált megoldás a csúcspontok szülő függvényének segítségével határozható meg: az **A-C-D-E út**. Ennek a megoldásnak a költsége 16, ez **nem optimális** megoldás.

Megoldás A* algoritmussal

Nyílt halmaz (perem)	Zárt halmaz
A(NULL, 0, 5)	
B(A, 6, 13), C(A, 2, 8)	A(NULL, 0, 5)
B(A, 6, 13), D(C, 6, 11)	A(NULL, 0, 5), C(A, 2, 8)
B(A, 6, 13), E(D, 16, 16)	A(NULL, 0, 5), C(A, 2, 8), D(C, 6, 11)
E(B, 14, 14)	A(NULL, 0, 5), B(A, 6, 13), C(A, 2, 8), D(C, 6, 11)
	A(NULL, 0, 5), B(A, 6, 13), C(A, 2, 8), D(C, 6, 11), E(B, 14, 14)

A megtalált megoldás a csúcspontok szülő függvényének segítségével határozható meg: az **A-B-E út**. Ennek a megoldásnak a költsége 14, ez **optimális** megoldás.

2. feladat



Kezdőállapot: A, célállapot: E, a szögletes zárójelben levő értékek a heurisztika értékek.

Megoldás mohó legjobbat-először kereséssel

Nyílt halmaz (perem)	Zárt halmaz
A(NULL, 0, 7)	
B(A, 3, 5), D(A, 7, 6)	A(NULL, 0, 7)
C(B, 6, 4), D(A, 7, 6)	A(NULL, 0, 7), B(A, 3, 5)
D(A, 7, 6), E(C, 17, 0)	A(NULL, 0, 7), B(A, 3, 5), C(B, 6, 4)
D(A, 7, 6)	A(NULL, 0, 7), B(A, 3, 5), C(B, 6, 4), E(C, 17, 0)

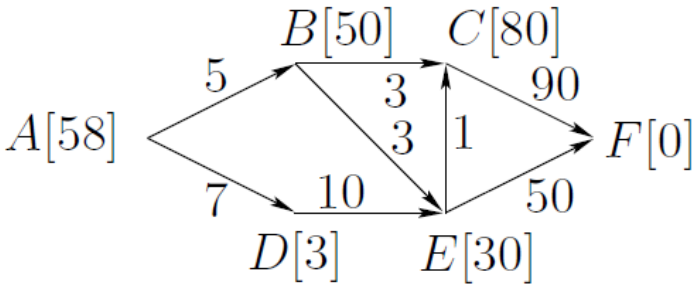
A megtalált megoldás az **A-B-C-E út**. Ennek a megoldásnak a költsége 17, ez **nem optimális** megoldás.

Megoldás A* algoritmussal

Nyílt halmaz (perem)	Zárt halmaz
A(NULL, 0, 7)	
B(A, 3, 8), D(A, 7, 13)	A(NULL, 0, 7)
C(B, 6, 10), D(A, 7, 13)	A(NULL, 0, 7), B(A, 3, 8)
D(A, 7, 13), E(C, 17, 17)	A(NULL, 0, 7), B(A, 3, 8), C(B, 6, 10)
E(D, 14, 14)	A(NULL, 0, 7), B(A, 3, 8), C(B, 6, 10), D(A, 7, 13)
	A(NULL, 0, 7), B(A, 3, 8), C(B, 6, 10), D(A, 7, 13), E(D, 14, 14)

A megtalált megoldás az **A-D-E út**. Ennek a megoldásnak a költsége 14, ez **optimális** megoldás.

3. feladat (megoldás nélkül)



Kezdőállapot: A, célállapot: F, a szögletes zárójelben levő értékek a heurisztika értékek.

5. óra - Játékok

A hagyományos MI egyik fő érdeklődési területe, elsősorban a sakk motiválta.

Még mindig érdekes terület, pl. a go játék még megoldatlan.

Kétszemélyes, lépésváltásos, determinisztikus, zéró összegű játék

Az általunk vizsgált játékokról feltesszük, hogy kétszemélyesek, lépésváltásosak, determinisztikusak és zéró összegűek.

Hasonló az állapottérben való kereséshez, azonban vannak lényeges különbségek. A problémák modellezése a következő elemekkel történik:

- **lehetséges állapotok halmaza** (legális játékállások)
- **kezdőállapot**
- **állapotátmenet függvény**, amely minden állapothoz hozzárendel egy (cselekvés,állapot) típusú rendezett párokból álló halmazt
- **végállapotok** (vagy célállapotok) halmaza (lehetséges állapotok részhalmaza)
- **hasznosságfüggvény**, amely minden lehetséges végállapothoz hasznosságot rendel.

De: itt **két ágens** van, felváltva lépnek (azaz alkalmaznak operátort), és a hasznosságfüggvényt az egyik maximalizálni akarja (MAX játékos), a másik minimalizálni (MIN játékos). Konvenció szerint **MAX kezd**.

Az első végállapot elérésekor a játéknak definíció szerint vége.

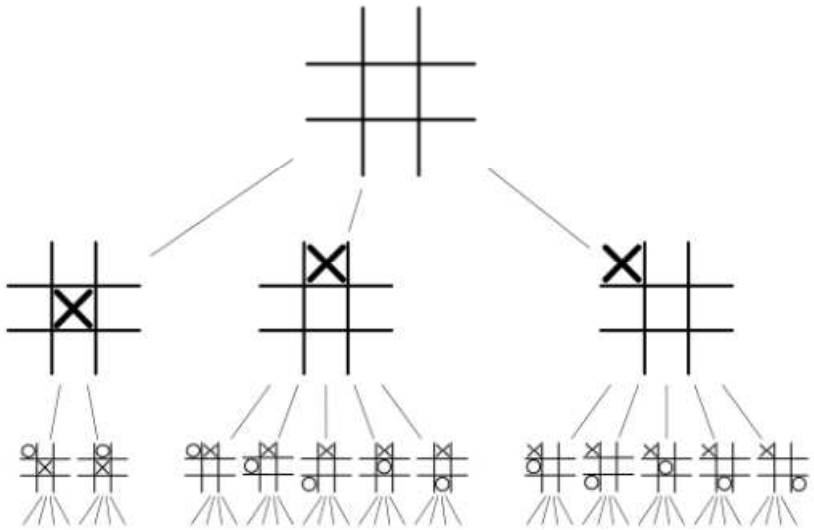
Zéró összegű játék: amennyit MAX nyer, pont annyit veszít MIN. Modellünkben MIN minimalizálja a hasznosságot, ami ugyanaz, mint maximalizálni a negatív hasznosságot, tehát MIN számára a hasznosság vehető MAX hasznossága mínusz egyszeresének, tehát a fenti modell zéró összegű.

A problémák modellje egy irányított gráfot definiál, melynek neve **játékgráf** (általában nem fa).

Példa

A 3x3-as amőba (tic-tac-toe) modellezése:

- lehetséges állapotok halmaza: lehetséges játékállások
- kezdőállapot: az üres tábla
- lehetséges cselekvések halmaza: a soron lévő játékos a saját jelét rakja valamelyik üres mezőre (MAX játékos az X jelet használja és ő kezd)
- végállapotok: amikor három azonos jel van egy sorban, oszlopban vagy átlóban, vagy tele van a tábla
- hasznosságfüggvény: értéke 1, ha X-ből jön ki a három, -1, ha körből, 0, ha egyikből sem



255168 lehetséges játék, 138 lehetséges kimenetel (végállapot).

Minimax algoritmus

Tegyük fel, hogy mindkét játékos a teljes játékgráfot ismeri, tetszőlegesen komplex számításokat tud elvégezni, és nem hibázik (nagyon erős feltevések). Ezt szokás **a tökéletes racionalitás hipotézisének** nevezni.

Egy stratégia minden állapotra meghatározza, hogy melyik lépést kell választani. Belátható, hogy mindkét játékos számára a lehető legjobb stratégiát a **minimax algoritmus** alapján lehet megvalósítani tökéletes racionalitás esetén.

A minimax algoritmus a következő értékeket számolja ki minden n csúcsra:

$$\text{minimax}(n) = \begin{cases} \text{hasznosság}(n) & \text{ha végállapot} \\ \max_{a\ n\text{ szomszédja}} \text{minimax}(a) & \text{ha MAX jön} \\ \min_{a\ n\text{ szomszédja}} \text{minimax}(a) & \text{ha MIN jön} \end{cases}$$

Az algoritmus:

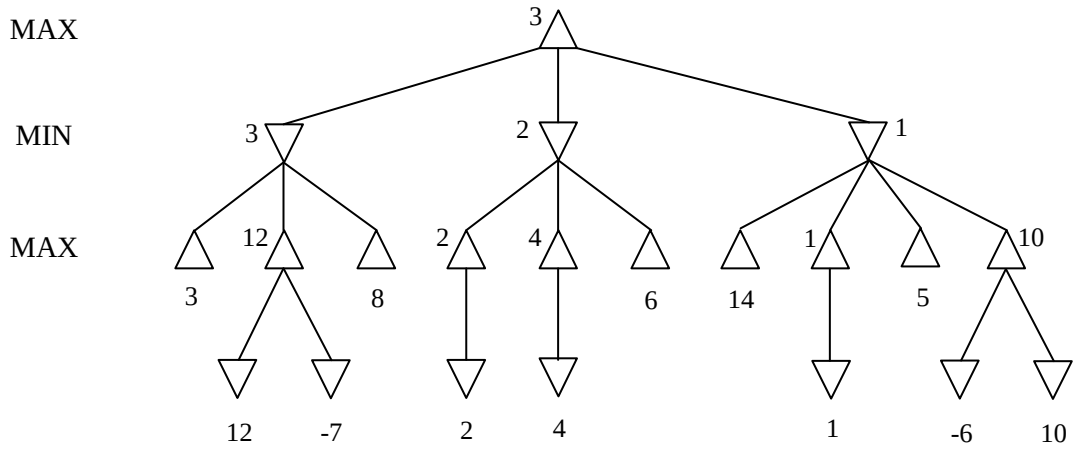
```
maxÉrték(n)
1 if végállapot(n)
  return hasznosság(n)
2 max <- -végtelen
3 for a in n szomszédai
4   max = max(max, minÉrték(a))
5 return max
```

```
minÉrték(n)
1 if végállapot(n)
  return hasznosság(n)
2 min <- +végtelen
3 for a in n szomszédai
4   min = min(min, maxÉrték(a))
5 return min
```

Az algoritmus a `maxÉrték(kezdőcsomópont)` hívással indít, feltéve hogy MAX játékos következik.

Világos, hogy a minimax érték az optimális hasznosság, amit az adott állapotból egy játékos elérhet, ha az ellenfél tökéletesen racionális.

1. feladat



Az ábrán a levél csomópontok a végállapotok, az alattuk szereplő értékek a hozzájuk tartozó hasznosságértékek (amik így egyben minimax értékek is), a nem levél csomópontok mellett szereplő értékek pedig a minimax algoritmus által a csomópontokhoz rendelt minimax értékek.

Futtatás: a játékos tehát először kiszámítja a minimax értékeket a teljes játékfára, majd mindig a számára legjobb minimax értékű szomszédot lépi (ez határozza meg a stratégiát). Ez csak elméleti jelentőségű, a minimax algoritmus nem skálázódik. Sakkban pl. 10^{154} csúcs a játékfában, 10^{40} különböző. A világegyetem atomjainak száma: kb. 10^{80}. Hogyan lehet minimaxot hatékonyan számolni? És/vagy közelítőleg? (Egyáltalán minimaxot kell mindig számolni?)

Időigény: $O(b^m)$

Ha a jatékgráfban van kör, akkor nem terminál (fakeresés), de a gyakorlatban ez nem probléma: csak fix mélységig futtatjuk (l. később) ill. a játékszabályok gyakran kizárják a végtelen köröket (ilyenkor a játékgráfban nincs kör).

Alfa-béta vágás

A minimax keresés problémája, hogy a játékban a megvizsgálandó állapotok száma exponenciális a lépések számában. A kitevőtől sajnos megszabadulni nem tudunk, ám lényegében megfelezhetjük.

Ötlet: ha tudjuk, hogy pl. MAX már ismer legalább egy olyan stratégiát, amellyel ki tud kényszeríteni pl. legalább 10 értékű hasznosságot egy adott állapotból indulva, akkor az adott állapot alatt nem kell vizsgálni olyan állapotokat, amelyekben MIN ki tud kényszeríteni ≤ 10 hasznosságot, hiszen tudjuk, hogy MAX sosem fogja ide engedni a játékot.

Ehhez az eddigi n paraméter mellé az alfa és béta új paramétereket is átadjuk az algoritmusnak.

Alfa: biztos, hogy MAX ki tudja kényszeríteni, hogy a hasznosság legalább alfa lesz, valamely olyan állapotból indulva, ami n állapotból a gyökér felé vezető úton van. Azt a minimum értéket jelenti, amit MAX már biztosított magának.

Béta: biztos, hogy MIN ki tudja kényszeríteni, hogy a hasznosság legfeljebb béta lesz, valamely olyan állapotból indulva, ami n állapotból a gyökér felé vezető úton van. Azt a maximum értéket jelenti, amit a MIN játékos biztosított magának.

Az algoritmus:

```
maxÉrték(n, alfa, béta)
1 if végállapot(n) return
  hasznosság(n)
2 max <- -végtelen
3 for a in n szomszédai
4   max = max(max, minÉrték(a,
      alfa, béta))
5   if max>=beta return max
6   alfa = max(max, alfa)
7 return max
```

```
minÉrték(n, alfa, béta)
1 if végállapot(n) return
  hasznosság(n)
2 min <- +végtelen
3 for a in n szomszédai
4   min = min(min, maxÉrték(a,
      alfa, beta))
5   if alfa>=min return min
6   beta = min(min, beta)
7 return min
```

Az algoritmus a `maxÉrték(kezdőcsomópont, -végtelen, +végtelen)` hívással indít, feltéve hogy MAX játékos következik, tehát a kezdő csomópont alfa és béta értékét rendre mínusz végtelenre illetve plusz végtelenre inicializálja. Ezt követően a fa rekurzív vizsgálata közben folyamatosan állítgatja az alfa és béta értékeket a két játékos által garantáltan elérhető értékekre. Ez oly módon történik, hogy a csomópontok kezdetben öröklik az alfa és béta értékeket a szülőjüktől, majd a gyerekeik vizsgálata során ha MAX csomópontban a gyerek minimax értéke nagyobb, mint az aktuális alfa, akkor az alfát frissítjük a gyerek minimax értékére, míg ha MIN csomópontban a gyerek minimax értéke kisebb, mint az aktuális béta, akkor a bétát frissítjük a gyerek minimax értékére. Ha MAX játékos esetén az aktuális minimax érték nagyobb vagy egyenlő mint az aktuális béta, vagy ha MIN játékos esetén az aktuális minimax érték kisebb vagy egyenlő mint az aktuális alfa, akkor az azt jelenti, hogy ez az állás – ha mind a két játékos részéről a legjobb játékot tételezzük fel – nem állhat elő, így nincs is értelme tovább vizsgálni, és az adott csomópont követői levághatók. (Ezt a vágást MAX játékos esetén béta-vágásnak, MIN játékos esetén alfa-vágásnak nevezzük.)

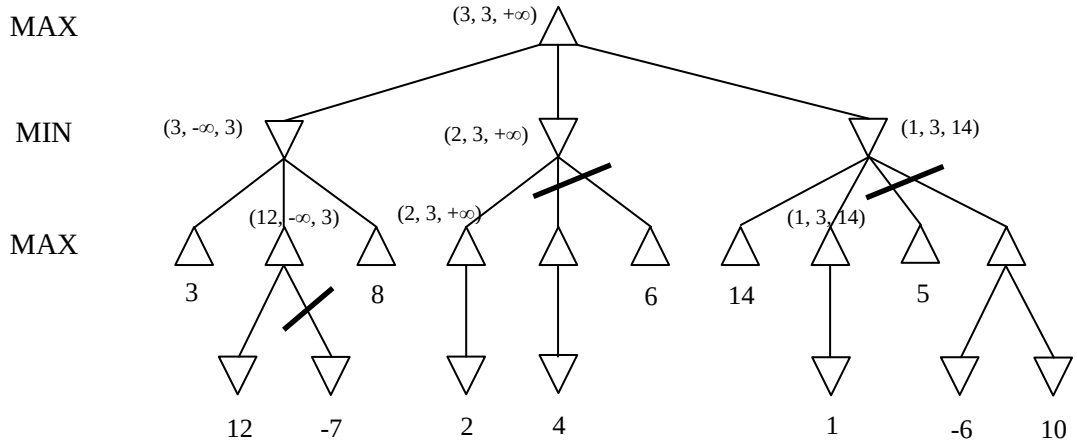
Időigény: erősen függ a követő csomópontok vizsgálatának sorrendjétől.

- ha mindig a legjobb csúcsot tudnánk választani kifejtésre (optimális eset), akkor $O(b^{m/2})$
- véletlen bejárással $O(b^{3m/4})$

A gyakorlatban használhatunk rendezési heurisztikákat, amik sokszor közel kerülnek az optimális esethez.

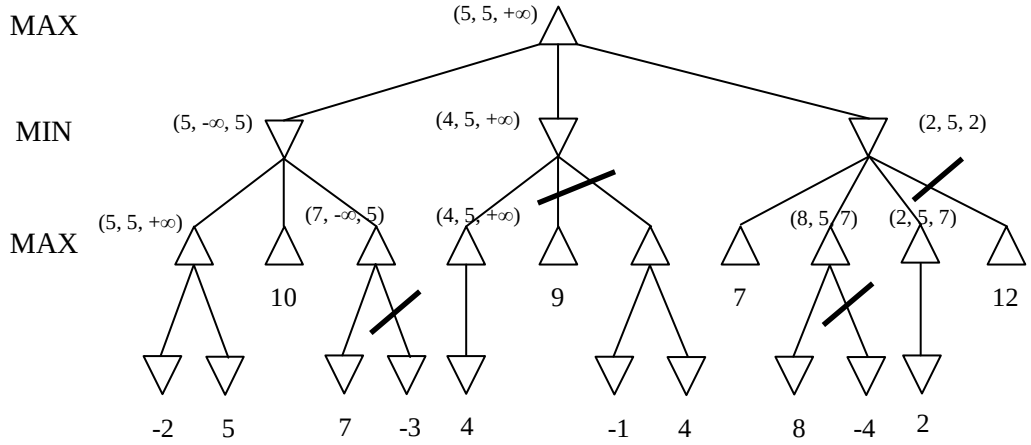
Gráf keresés: játékgráfban is sok lehet az ismétlődő állapot. Eddig a fakeresés analógiájára nem tároltuk a már látott állapotokat. Ha tároljuk (mint a zárt halmazban) akkor az alfa-béta algoritmus is jelentősen felgyorsul, ill. nem ragad végtelen ciklusba. Itt a hagyományos elnevezése a zárt halmaznak transzpozíciós tábla.

1. feladat



Az ábrán a levél csomópontok a végállapotok, az alattuk szereplő értékek a hozzájuk tartozó hasznosságértékek (amik így egyben minimax értékek is). Az algoritmus a játékgráf csomópontjait (a rekurzív függvényhívások miatt) a mélységi keresésnek megfelelő sorrendben járja be, a csomópontokhoz tartozó minimax, alfa és béta értékeket közben folyamatosan frissítve. A nem levél csomópontok mellett szereplő értékek az algoritmus teljes futása után a csomópontokhoz rendelt (minimax, alfa, béta) hármasok.

2. feladat



Praktikus, valós idejű algoritmusok

A minimax algoritmus a játék egész keresési terét állítja elő, az alfa-béta nyesés viszont lehetőséget ad annak nagy részét lenyesni. Az alfa-béta nyesésnek mégis a végállapotokig kell keresnie legalább a keresési tér egy részében. Nem tudunk a végállapotokig leérni: heurisztikus kiértékelő függvények kellenek, amik bármely állapot minőségét jellemzik, és az alfa-béta algoritmus korábban kell, hogy visszalépjen, pl. fix mélységből, vagy még okosabban.

Heurisztikus értékelés

A heurisztika területspecifikus tudást ad (mint korábban az A* algoritmusnál) az egyébként területfüggetlen kereső algoritmushoz. A heurisztikus kiértékelésre teljesülnie kell, hogy:

- a végállapotokra az eredeti értéket adja
- gyorsan kiszámolható
- nem-végállapotokra a nyerési esélyt jól tükröző érték (finomabb felbontás, mint a minimax értékek, ami gyakran 1/-1).

Hogyan tervezhetünk ilyet?

- Lineáris jellemzőkombinációk
- Nemlineáris jellemzőkombinációk
- Tanulás

Keresés levágása

Hogyan használjuk ki a heurisztikus értékelést? Több lehetőség:

- **Fix mélység**: az aktuális játékállásból pl. X mélységig építjük fel a keresőfát alfa-bétával, ami során az X mélységű állapotokat értékeljük ki heurisztikusan. Az implementáció egyszerű: u.a., csak a mélységet is követni kell, és végállapotok mellett az X mélységre is tesztelünk a rekurzió előtt.
- **Iteratívan mélyülő keresés**: alfa-béta iteráltan növekvő mélységig. Ez analóg a korábbi iteratívan mélyülő kereséssel (az alfa-béta pedig a mélységi kereséssel). Ennek a legnagyobb előnye, hogy rugalmas: már nagyon gyorsan van tippünk következő lépésre, és ha van idő, akkor egyre javul. Valós időben használható tehát.
- **Horizont effektus miatti javítások**: fix keresési mélységnél lehet, hogy egy fontos következmény (pl. ütés) nem lesz figyelembe véve, mert picit lentebb van. Technikák okosabb vágásra:
    - Egyensúlyi keresés (quiescence search): ha egy állapot „mozgalmas” (heurisztika dönt, hogy ez mikor van, de pl. ha az úton a heurisztikus értékelés nagyon változik (pl. sok ütés)), akkor lentebb megyünk, addig, amíg „megnyugszik” az adott lépésvariáció, azaz várható, hogy ha még lentebb mennénk, akkor viszonylag sokáig stabil lenne.
    - Szinguláris kiterjesztés: az olyan állapotokból, ahol van „világosan legjobb” lépés (ezt is heurisztika dönti el), ott nem állunk meg a maximális mélységben, viszont csak a legjobb lépést terjesztjük ki. Így érdekesebb variációkat elég mélyen meg lehet nézni, mert az elágazási faktor 1 ezekben.

Véletlent tartalmazó játékok

Sok játékban van véletlen, pl. kockadobás. Olyan eseteket nézünk, ahol az állapot továbbra is teljesen ismert, de a véletlentől is függ. Egészítsük ki a játékfát a véletlen eseményt leíró csúcsokkal (CHANCE), mintha a véletlen lenne a harmadik játékos.

A minimax algoritmust nem tudjuk direktben alkalmazni, mert a CHANCE „játékos”-ra nem érvényes, hogy optimalizálna, csak véletlenül választ függetlenül a következményektől. Módosítás: várható-minimax, ami a minimax várható értéket adja meg, tehát a CHACE csúcsokban valószínűséggel súlyozott átlagot kell számolni.

Alfa-béta is implementálható, a MIN és MAX csúcsokon változatlan formában, sőt, a CHANCE csúcsokat is lehet vágni, ha a várható értéket tudjuk korlátozni a lehetséges kimenetelek egy részhalmazának a vizsgálata után. Ez pedig csak akkor lehetséges, ha a hasznosságfüggvény korlátos.

Ha nem teljes az információ

Van, hogy a játék állapota nem ismert teljesen, pl. kártyajátékok. Ez mindig egy véletlen esemény (pl. osztás) eredménye. Nem célravezető azonban egyszerűen várható értéket venni, mert nem mindegy, hogy „most még nem tudom mi lesz, de később majd meglátom” ill. „nem tudom mi lesz, és később se fogom pontosan megtudni”. Pl. a kockázat máshogy jelentkezik (nagyobb az ismeretlen állapotok esetében).

Itt lehet pl. az állapotokra vonatkozó „hiedelmek” terében keresni, stb.

6. óra - Felügyelt tanulás

A tanulás

Egy tanuló ágens felfogható úgy, mint aminek egy cselekvő és egy tanuló komponense van. A **cselekvő komponens** eldönti, hogy az ágens egy adott helyzetben mit cselekedjen, a **tanuló komponens** pedig módosítja a cselekvő komponenst annak érdekében, hogy az a jövőben jobb döntéseket hozzon.

A tanulás lényege: tapasztalati (megfigyelt) tények felhasználása arra, hogy egy racionális ágens teljesítményét növeljük.

A tanulás típusai:

- **ellenőrzött/felügyelt tanulás** (supervised learning**):** egy leképezésnek a bemeneti és kimeneti minták alapján történő megtanulását jelenti. Tehát a feladat egy f függvény tanulása (x_i, $f(x_i)$) minták alapján. Példa: kézzel osztályozott email-ek alapján a spam és nem spam email-ek tanulása.
- **nem ellenőrzött/felügyelet nélküli tanulás** (unsupervised learning): bemeneti minták tanulása történik, de a kívánt kimeneti minták nem biztosítottak. Tehát a feladat példák osztályozásának tanulása pusztán a példák (x_i) ismeretében. Példa: email-ek különböző csoportokba sorolása automatikusan téma szerint úgy, hogy a témakörök nem adottak (az algoritmus sem fogja megmondani a témakörök nevét).
- **megerősítéses tanulás** (reinforcement learning): egy leképezés tanulását jelenti oly módon, hogy a bemeneti és kimeneti minták nem adottak direkt módon. Tehát nem egy cselekvéshez (bemeneti mintához) adott a tanulandó függvény értéke, hanem egy adott cselekvéssorozathoz tartozik valamilyen, a cselekvéssorozat végén elért célállapotban értelmezett jutalom (megerősítés), és ez alapján kell a függvényt (optimális stratégiát) megtanulni. Példa: sakk játék tanulása egymás után lejátszott sok véletlen meccs alapján.

A felügyelt tanulás

A felügyelt tanulás esetén a példákhoz meg vannak adva a helyes kimeneti értékek is. A feladat egy f függvény tanulása (x_i, $f(x_i)$) minták alapján, tehát a még nem ismert példákhoz a hozzájuk tartozó függvényérték megmondása. A mintákat jellemzőkkel (feature) írhatjuk le. A különböző minták jellemzőinek és függvényértékeinek korrelációjából következtethetünk egy még ismeretlen minta függvényértékére. A jellemzők és a függvényértékek lehetnek binárisak (igen/nem), diszkrét értékűek, valamint valós értékűek.

Felügyelt tanulás fajtái:

- **Osztályozás:** a tanulandó függvény értékkészlete diszkrét
  - **Egyosztályos tanulás**
  - **Kétosztályos tanulás**
  - **Többosztályos tanulás**
- **Regresszió:** a tanulandó függvény értékkészlete valós

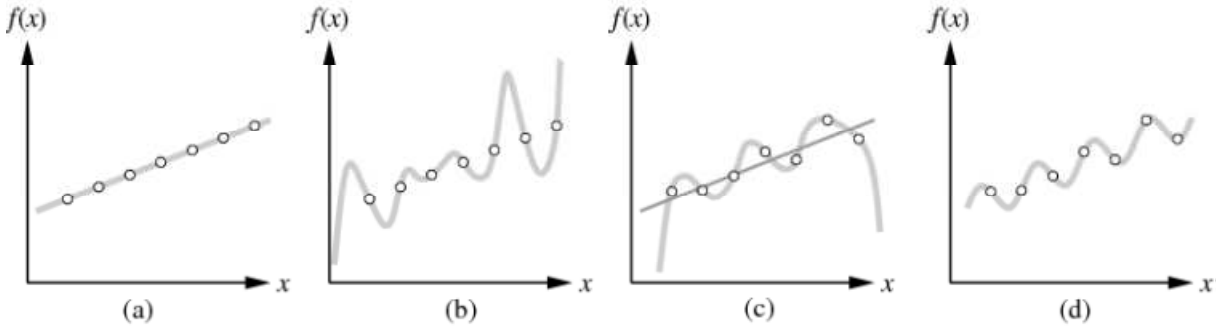
Az **induktív következtetés** feladata egy olyan h **hipotézis** függvény keresése egy f **tanulandó** függvényre vonatkozó minták halmaza alapján, amely a lehető legjobban közelíti az f függvényt.

A h függvény **konzisztens** az adatokkal ha h(x) = f(x) minden tanító példára. A gyakorlatban sokszor elég ha h „közel" van a példákhoz, nem feltétlenül kell pontosan illeszkednie.

A h függvény mindig egy H hipotézistér eleme. H lehet pl. a legfeljebb k-ad fokú polinomok halmaza, vagy bármilyen más alkalmas függvényhalmaz. A tanulás **realizálható** ha f ∈ H.

Egy adott f függvényhez sok olyan h függvény lehet, ami jól közelíti a mintákat/konzisztens a mintákkal. Melyiket válasszuk? **Indukció problémája**: f jó közelítése olyan amely a példákon kívül is jól közelíti f-et, azaz jól általánosít. Ez nehéz és nagyon érdekes kérdés!

Pl. az a h, amelyre h(x) = f(x) minden példára, egyébként h(x) = 0, tökéletesen megtanulja a példákat, de a lehető legrosszabban általánosít (általában...). Ez a **magolás** (rote learning). Emiatt tömör/egyszerű reprezentációra kell törekedni, lehetőleg tömörebbre mint a példák listája, így biztosan kiküszöböljük a magolást. Ez az elv az **Occam borotvája**: ha más alapján nem tudunk választani, akkor a lehető legtömörebb/legegyszerűbb leírást kell venni. Tehát részesítsük előnyben a legegyszerűbb olyan hipotézist, amely konzisztens az adatokkal / jól közelíti az adatokat.



Az (a)-beli adatpontokra (a)-t érdemes illeszteni (b) helyett, mert (a) sokkal egyszerűbb, így sokkal valószínűbb, hogy jól általánosít.

Ha hipotézistérnek a legfeljebb k-ad fokú polinomok halmazát választjuk, akkor (c) adatpontjaira inkább megéri egy nem konzisztens egyenest illeszteni, mint egy sokadfokú polinomot, mert valószínűtlen, hogy a sokadfokú polinom általánosítana jól.

Fontos észben tartani, hogy a hipotézistér megválasztásán is sok múlik, ugyanis a (d), egy egyszerű ax + b + c sin(x) alakú függvény, pontosan illeszkedik a (c)-beli adatpontokra, és így ez a legegyszerűbb konzisztens hipotézis, azonban ez nincs benne a legfeljebb k-ad fokú polinomok halmazában.

A tanuló algoritmus teljesítményének becslése

A kiértékelés menete

Akkor tudjuk azt eldönteni, hogy egy adott algoritmus mennyire általánosít jól, ha egy a **tanítóhalmaztól teljesen független teszthalmazon** végezzük az algoritmus kiértékelését. Ezzel kiszűrhető a magolás. Ha a tanító halmazon tesztelnénk, akkor az az algoritmus teljesítene a legjobban, ami pusztán bemagolja az adatokat.

Kukucskálás (peeking): Hiába szeparáljuk a teszthalmazt; ha a teszthalmazon látott teljesítmény alapján optimalizáljuk az algoritmusunk paramétereit, akkor a teszthalmaz is hatással lesz az eredményre, és onnantól fogva nem tudjuk, hogy az algoritmus milyen jól általánosít. Ezért elvben minden paraméter-beállításnál új teszthalmaz kellene. Mivel a gyakorlatban ez általában megvalósíthatatlan, ezért az adatokat 3 független részhalmazra szokás osztani. Az első részhalmazon **(tanító halmaz**) tanítjuk az algoritmust, a második részhalmazon optimalizáljuk az algoritmus paramétereit (**fejlesztési/development halmaz**), míg a harmadik részhalmazon végezzük el a végső algoritmus tesztelését (**teszthalmaz**). Ily módon a kukucskálás kiszűrhető és megtudhatjuk, hogy az algoritmusunk mennyire jól általánosít.

A hipotézisek kiértékelése

Bináris osztályozás esetén

		Tényleges osztály	
		+	-
Predikált osztály	+	True positive (TP)	False positive (FP)
	-	False negative (FN)	True negative (TN)

$$\text{Helyesség (Accuracy)} = \frac{TP + TN}{TP + TN + FP + FN}$$

$$\text{Hiba (Error)} = \frac{FP + FN}{TP + TN + FP + FN}$$

$$\text{Pontosság (Precision)} = \frac{TP}{TP + FP}$$

$$\text{Fedés (Recall)} = \frac{TP}{TP + FN}$$

$$\mathrm{F}_1 = 2 * \frac{Pontosság * Fedés}{Pontosság + Fedés}$$

$$\mathrm{F}_\beta = (1 + \beta^2) * \frac{Pontosság * Fedés}{(\beta^2 * Pontosság) + Fedés}$$

Általános osztályozás esetén

$$\text{Helyesség (Accuracy)} = \frac{\#\ helyesen\ felismert\ példa}{\#\ összes\ példa}$$

$$\text{Hiba (Error)} = \frac{\#\ hibásan\ felismert\ példa}{\#\ összes\ példa}$$

Regresszió esetén

$$\text{Négyzetes hiba (Squared error)} = \sum_{i=1}^{n}\big(h(x_i) - f(x_i)\big)^2$$

$$\text{Átlagos négyzetes hiba (Mean squared error)} = \frac{1}{n}\sum_{i=1}^{n}\big(h(x_i) - f(x_i)\big)^2$$

$$\text{Középhiba (Root} - \text{mean} - \text{square error)} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\big(h(x_i) - f(x_i)\big)^2}$$

$$\text{Pearson} - \text{féle korrelációs együttható (Pearson's correlation coefficient)} = \frac{\sum_{i=1}^{n}\big(h(x_i) - \overline{h(x)}\big)\big(f(x_i) - \overline{f(x)}\big)}{\sqrt{\sum_{i=1}^{n}\big(h(x_i) - \overline{h(x)}\big)^2}\sqrt{\sum_{i=1}^{n}\big(f(x_i) - \overline{f(x)}\big)^2}}$$

Spearman-féle rangkorrelációs együttható (Spearman's rank correlation coefficient) = *a Pearson-féle korrelációs együttható olyan speciális esete, ami a numerikus értékek helyett azok rangjával számol*

Példányalapú tanulás

Ötlet: egy pont tulajdonságai valószínűleg hasonlók az adott pont környezetébe eső pontok tulajdonságaihoz, ezért adott x-re az x-hez "közeli" tanító példák alapján határozzuk meg h(x) értékét.

k-legközelebbi-szomszéd (kNN)

Az algoritmus:

1. Keressük meg az x_i pont k db legközelebbi szomszédját
2. A $h(x_i)$ értékét határozzuk meg a k db legközelebbi szomszéd f(x) értéke alapján
    a. osztályozás esetén pl. a szomszédok f(x) értékeinek többségi szavazata: $h(x_i) = \text{TSZ}_{j=1}^{\text{k}}\left(f(x_j)\right)$
    b. regresszió esetén pl. a szomszédok f(x) értékeinek átlaga: $h(x_i) = \frac{1}{k}\sum_{j=1}^{k} f(x_j)$

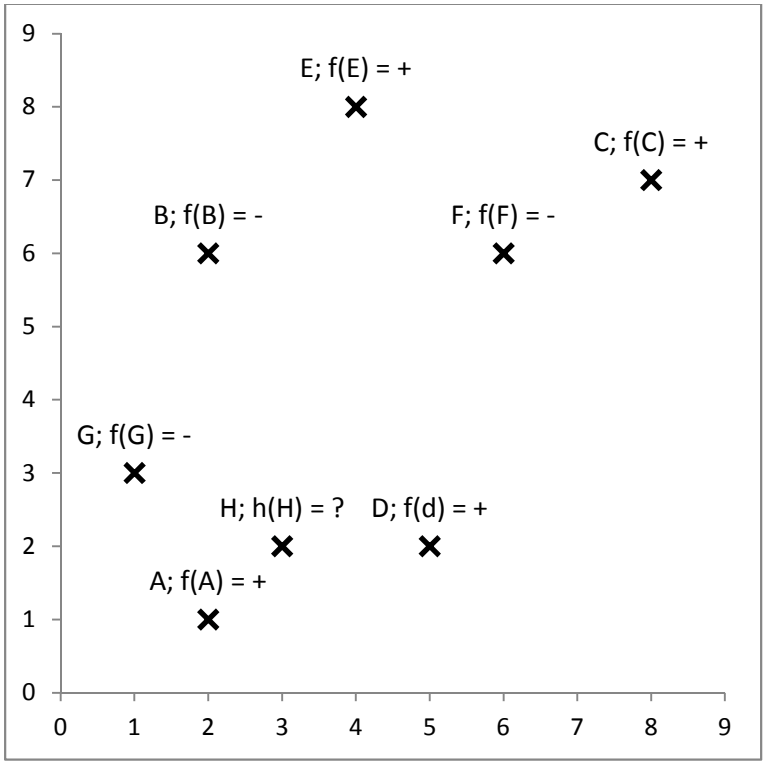
A legközelebbi szomszédok meghatározásához szükséges a **távolságfüggvény** definiálása. Ez egy dist(x_i,x_j) függvény, ahol x_i és x_j egy-egy jellemzővektorral van megadva. Ez sokféle lehet, pl.:

- Euklideszi távolság: $dist_E(x_i, x_j) = \sqrt{\sum_{l=1}^{m}(x_{i,l} - x_{j,l})^2}$
- Manhattan távolság: $dist_M(x_i, x_j) = \sum_{l=1}^{m}|x_{i,l} - x_{j,l}|$

Egyszerű, kevés hangolást igényel, nem kell külön modellt építeni, és sokszor jó algoritmus. De vannak hibái:

- érzékeny a távolságfüggvény definíciójára,
- sok példa esetén költséges a k legközelebbi szomszédot megtalálni (bár vannak algoritmusok és adatszerkezetek amik segítenek, de ez külön kutatási terület),
- ha sok jellemző van (sokdimenziós a tér) akkor „mindenki távol van mindenkitől”.

1. feladat



Feladat: a H címkéjének (+/-) predikálása az A-G tanító példák címkéje alapján (osztályozás). Távolságfüggvényként használjuk az Euklideszi távolságot.

1. H és a tanító példák közti távolság meghatározása:

$$dist_E(H, A) = \sqrt{(3-2)^2 + (2-1)^2} = \sqrt{2} \approx 1{,}414$$

$$dist_E(H, B) = \sqrt{(3-2)^2 + (2-6)^2} = \sqrt{17} \approx 4{,}123$$

$$dist_E(H, C) = \sqrt{(3-8)^2 + (2-7)^2} = \sqrt{50} \approx 7{,}071$$

$dist_E(H, D) = \sqrt{(3-5)^2 + (2-2)^2} = \sqrt{4} = 2$

$dist_E(H, E) = \sqrt{(3-4)^2 + (2-8)^2} = \sqrt{37} \approx 6{,}083$

$dist_E(H, F) = \sqrt{(3-6)^2 + (2-6)^2} = \sqrt{25} = 5$

$dist_E(H, G) = \sqrt{(3-1)^2 + (2-3)^2} = \sqrt{5} \approx 2{,}236$

2. Következtetés a k db legközelebbi szomszéd alapján:

k = 3 => $h(H) = \text{TSZ}\big(f(A), f(D), f(G)\big) = TSZ(+, +, -) = +$

k = 5 => $h(H) = \text{TSZ}\big(f(A), f(D), f(G), f(B), f(F)\big) = TSZ(+, +, -, -, -) = -$

k = 7 => $h(H) = \text{TSZ}\big(f(A), f(D), f(G), f(B), f(F), f(E), f(C)\big) =$

$= \text{TSZ}(+, +, -, -, -, +, +) = +$

Kernelmódszer

A kernelmodellben úgy tekintünk minden tanító példányra, mintha egy kis saját sűrűségfüggvényt - **kernelfüggvényt** (kernel function) - generálna. Egy x_i tanító példa egy $K(x, x_i)$ kernelfüggvényt generál, amely a tér minden x pontjához egy valószínűséget rendel.

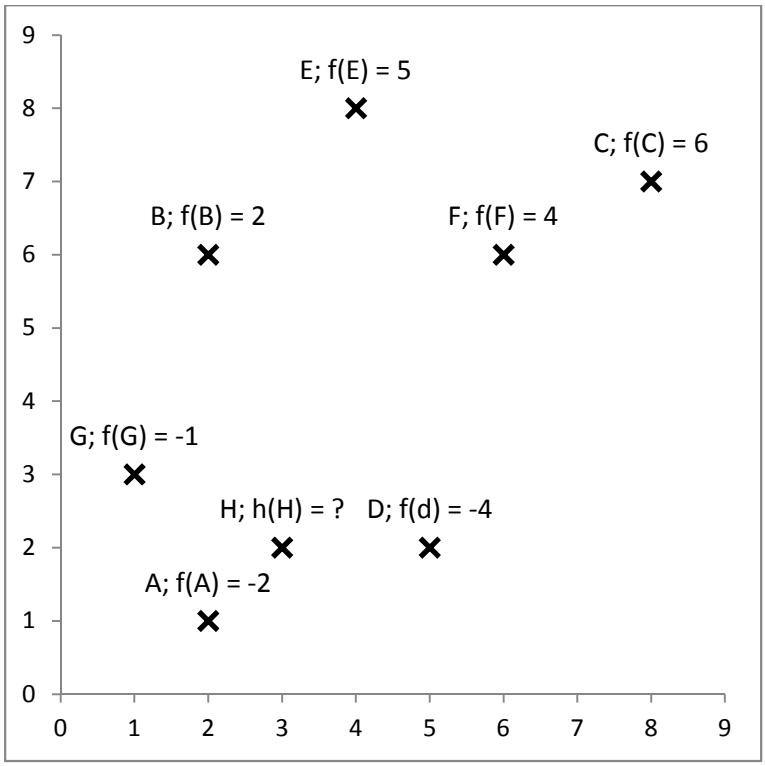
Normális esetben a kernelfüggvény csak az x és x_i közötti $dist(x, x_i)$ távolságtól függ. Különféle kernelfüggvények használatosak, a legnépszerűbb a Gauss-függvény. Egy d dimenziós adatokon értelmezett gömbszimmetrikus Gauss-kernel w szórással:

$$K(x, x_i) = \frac{1}{(w^2\sqrt{2\pi})^d} e^{-\frac{dist_E(x, x_i)^2}{2w^2}}$$

A $h(x_i)$ értékének kiszámításában minden tanító példának szerepe van:

- osztályozás esetén pl. a tanító példák f(x) értékeinek kernelfüggvénnyel súlyozott többségi szavazata: $h(x_i) = \text{TSZ}_{j=1}^{n}\big(f(x_j),\ K(x_j, x_i)\ súlyokkal\big)$
- regresszió esetén pl. a tanító példák f(x) értékeinek kernelfüggvénnyel súlyozott átlaga: $h(x_i) = \frac{\sum_{j=1}^{n} K(x_j, x_i) f(x_j)}{\sum_{j=1}^{n} K(x_j, x_i)}$

1. feladat



Feladat: a H függvényértékének (egy valós szám) predikálása az A-G tanító példák függvényértéke alapján (regresszió). Kernelfüggvényként használjuk a Gauss-kernelt, w=1 szórással (d=2, mivel az adatok kétdimenziósak).

$$h(H) = \frac{K(A,H)f(A) + K(B,H)f(B) + K(C,H)f(C) + K(D,H)f(D)}{K(A,H) + K(B,H) + K(C,H) + K(D,H) + K(E,H) + K(F,H) + K(G,H)} +$$

$$+ \frac{K(E,H)f(E) + K(F,H)f(F) + K(G,H)f(G)}{K(A,H) + K(B,H) + K(C,H) + K(D,H) + K(E,H) + K(F,H) + K(G,H)} =$$

$$= \frac{0{,}059 * (-2) + 3{,}24 * 10^{-5} * 2 + 2{,}21 * 10^{-12} * 6 + 0{,}022 * (-4)}{0{,}059 + 3{,}24 * 10^{-5} + 2{,}21 * 10^{-12} + 0{,}022 + 1{,}47 * 10^{-9} + 5{,}93 * 10^{-7} + 0{,}013} +$$

$$+ \frac{1{,}47 * 10^{-9} * 5 + 5{,}93 * 10^{-7} * 4 + 0{,}013 * (-1)}{0{,}059 + 3{,}24 * 10^{-5} + 2{,}21 * 10^{-12} + 0{,}022 + 1{,}47 * 10^{-9} + 5{,}93 * 10^{-7} + 0{,}013} =$$

$$= -2{,}32$$

7. óra - Naiv-Bayes osztályozás

A naiv-Bayes osztályozás a felügyelt tanulás egy fajtája, tehát adottak bemeneti és kimeneti minták, és ezek alapján kell egy olyan h hipotézist találni, mely lehető legjobban közelíti az f függvényt. Általában a mintákat jellemzőkkel (feature) írhatjuk le. A különböző minták jellemzőinek és osztálycímkéinek korrelációjából következtethetünk egy még ismeretlen minta osztályára. A naiv-Bayes osztályozás először egy **valószínűségi modellt** épít fel a tanító példák alapján, majd egy ismeretlen minta osztálycímkéjét e modell alapján határozza meg. Mivel osztályozásról beszélünk, ezért az f függvény értékkészlete diszkrét.

Legyenek adottak az objektumokat leíró X_1, ... , X_n attribútum-változók és a C osztályváltozó. Ezen felül adott a mintáknak egy N elemű halmaza úgy, hogy minden mintánál ismert az attribútumok értéke és a minta osztálya. A feladat egy még ismeretlen, attribútumokkal leírt objektum besorolása a megfelelő osztályba az tanító példák alapján felépített valószínűségi modell segítségével.

A naiv-Bayes osztályozás során abba a $c^* \in C$ osztályba soroljuk az $x_1, \ldots, x_n$ jellemzőkkel leírt objektumot, melyre a $P(c \mid x_1, \ldots, x_n)$ valószínűség a legnagyobb, azaz

$$\mathrm{c}^* = \underset{\mathrm{c}\in\mathrm{C}}{\operatorname{argmax}}\ P(c \mid x_1, \ldots, x_n)$$

A probléma az, hogy a megfigyeléseinkből nem tudunk közvetlen $P(c \mid x_1, \ldots, x_n)$ valószínűséget becsülni, ezért próbáljuk a képletet olyan alakra hozni, hogy a benne szereplő valószínűségek könnyen megbecsülhetők legyenek. Első lépésként alkalmazhatjuk a **Bayes-szabályt**:

$$\mathrm{c}^* = \underset{\mathrm{c}\in\mathrm{C}}{\operatorname{argmax}}\ P(c \mid x_1, \ldots, x_n) = \underset{\mathrm{c}\in\mathrm{C}}{\operatorname{argmax}} \frac{P(x_1, \ldots, x_n \mid c) * P(c)}{P(x_1, \ldots, x_n)}$$

Sajnos általános esetben nem tudjuk tovább alakítani a képletet. Ahhoz, hogy ennél egyszerűbb alakra hozhassuk, **fel kell tegyük** azt, hogy az osztály ismeretében az **attribútumok egymástól feltételesen függetlenek**. Így a képlet a következő alakra hozható:

$$\mathrm{c}^* = \underset{\mathrm{c}\in\mathrm{C}}{\operatorname{argmax}} \frac{P(x_1, \ldots, x_n \mid c) * P(c)}{P(x_1, \ldots, x_n)} = \underset{\mathrm{c}\in\mathrm{C}}{\operatorname{argmax}} \frac{P(c) * \prod_{i=1}^{n} P(x_i \mid c)}{P(x_1, \ldots, x_n)}$$

Mivel azt a c osztályt keressük, ahol a képlet a maximális értéket veszi fel, de az itt felvett tényleges maximális érték lényegtelen, csak a maximum helye a fontos, ezért a maximális hely megkeresése szempontjából konstans adattagokkal (vagyis azokkal, amik nem tartalmazzák c-t) egyszerűsíthetünk. Így a naiv-Bayes osztályozás során a következő képletet alkalmazzuk:

$$\mathrm{c}^* = \underset{\mathrm{c}\in\mathrm{C}}{\operatorname{argmax}}\ P(c) \prod_{i=1}^{n} P(x_i \mid c)$$

És ez az alak azért lesz jó, mert a $P(c)$ illetve a $P(x_i \mid c)$ valószínűségek a megfigyelési adatokból általában könnyen becsülhetők (**empirikus valószínűség**):

$$P(c) \approx \frac{n_c}{N}$$

$$P(x_i \mid c) \approx \frac{n_{c,x_i}}{n_c}$$

ahol n_c azoknak a tanító mintáknak a száma, aminek az osztálya c, N a tanító minták száma, n_{c,x_i} pedig azoknak a tanító mintáknak a száma, ahol az osztály c és az X_i attribútum-változó értéke x_i.

Azonban a probléma a $P(x_i \mid c)$ kiszámításával az, hogy kevés minta esetén pontatlan becslést adhat, és egy nem megfigyelt attribútum-érték és osztály páros esetén ez a valószínűség akár 0 is lehet. Ilyen esetben az adott osztályhoz tartozó $P(c)\prod_{i=1}^{n} P(x_i \mid c)$ érték 0 lesz, függetlenül a többi valószínűségtől. Mivel a gyakorlatban általában egyetlen eseménynek sem 0 a valószínűsége, legfeljebb 0-hoz nagyon közeli, ezért ebben ilyenkor is valószínűleg pusztán a kevés tanító adat miatt 0 a becsült valószínűség, nem pedig azért, mivel az adott esemény soha nem fordulhat elő. Ezért a gyakorlatban célszerűbb ezeket a $P(x_i \mid c)$ valószínűségeket az **m-estimate módszerrel** közelíteni amely a 0 valószínűségeket egy 0-hoz közeli, de pozitív értékre korrigálja, és így pontosabb becslést ad:

$$P(x_i \mid c) \approx \frac{n_{c,x_i} + mp}{n_c + m}$$

ahol p az a priori (megfigyeléseket megelőző) valószínűsége az attribútumnak. Ez alapján $p = \frac{1}{k}$, ahol az X_i attribútum-változó k-féle értéket vehet fel. Az m pedig egy tetszőleges szám, ami azt mondja meg, hogy hány további véletlen mintát adunk hozzá a mintáink halmazához (ezek a minták a p valószínűség alapján lesznek generálva). Minél kevesebb a tanító mintánk, és így minél bizonytalanabbak vagyunk az empirikus valószínűségben, annál nagyobb m-et érdemes választani. Az m-estimate módszernek egy számos területen alkalmazott esete a **Laplace-simítás**, amikor $m = k$, vagyis

$$P(x_i \mid c) \approx \frac{n_{c,x_i} + 1}{n_c + k}$$

A $P(c)$ becsléséhez is lehetne az m-estimate módszert alkalmazni, de ez általában azért nem szokás, mivel elegendő a tanító példák száma ehhez a becsléshez. Az gyakorlaton az egyszerűség kedvéért a hagyományos empirikus becslést alkalmazzuk az összes valószínűség meghatározásához.

A bemutatott modell a naiv-Bayes modell/naiv-Bayes osztályozás. Azért naiv, mert feltételezi, hogy az osztály ismeretében az attribútumok egymástól feltételesen függetlenek, ami az esetek többségében nem igaz. Ennek ellenére nagyon hatékony és elég pontos. További előnye, hogy nagy méretű problémákra is jól alkalmazható, és nem jelentenek gondot számára a zajos adatok sem. Ezért gyakorlatban a naiv-Bayes osztályozók az egyik leggyakrabban alkalmazott és leghatékonyabb tanulási algoritmusok.

1. feladat

A táblázatban felsorolt megfigyelések alapján a naiv-Bayes modellt használva mi lesz a legvalószínűbb érdemjegye egy olyan hallgatónak a Mesterséges Intelligencia vizsgán, aki keveset jegyzetelt, hét napot tanult a vizsgára és gyakran járt a JATE-ba?

Jegyzetelés (Je)	Tanulás (T)	JATE (J)	Érdemjegy (É)
2	1	1	3
0	3	1	1
1	7	0	3
1	1	0	5
0	1	1	1
2	7	1	5
0	3	1	3

Jegyzetelés: 0=semmit sem jegyzetelt; 1=keveset jegyzetelt; 2=sokat jegyzetelt
Tanulás: 1=egy napot tanult; 3=három napot tanult; 7=hét napot tanult
JATE: 0=ritkán járt a JATE-ba, 1=gyakran járt a JATE-ba
Érdemjegy: a Mesterséges Intelligencia vizsga érdemjegye

É=1: $P(\text{É}=1) * P(Je=1 \mid \text{É}=1) * P(T=7 \mid \text{É}=1) * P(J=1 \mid \text{É}=1) = \frac{2}{7} * \frac{0}{2} * \frac{0}{2} * \frac{2}{2} = 0$

É=3: $P(\text{É}=3) * P(Je=1 \mid \text{É}=3) * P(T=7 \mid \text{É}=3) * P(J=1 \mid \text{É}=3) = \frac{3}{7} * \frac{1}{3} * \frac{1}{3} * \frac{2}{3} = \frac{2}{63}$

É=5: $P(\text{É}=5) * P(Je=1 \mid \text{É}=5) * P(T=7 \mid \text{É}=5) * P(J=1 \mid \text{É}=5) = \frac{2}{7} * \frac{1}{2} * \frac{1}{2} * \frac{1}{2} = \frac{2}{56}$

Mivel $0 < \frac{2}{63} < \frac{2}{56}$, ezért a táblázatban szereplő adatokon tanított naiv-Bayes modell alapján az a legvalószínűbb, hogy ötös érdemjegyet kap egy olyan hallgató a Mesterséges Intelligencia vizsgán, aki keveset jegyzetelt, hét napot tanult a vizsgára és gyakran volt a JATE-ban.

2. feladat

A táblázatban felsorolt megfigyelések alapján a naiv-Bayes modellt használva feltehetőleg késni fog-e a busz, ha épp esik az eső, van vásár a városban, nincsenek felújítási munkálatok az utakon és napközben várunk a buszra?

Eső (E)	Vásár (V)	Felújítás (F)	Időpont (I)	Késik (K)
+	+	+	3	+
-	+	+	2	+
+	-	+	1	+
-	+	-	3	+
-	-	+	2	+
+	+	-	2	+
+	+	-	3	+
+	+	+	3	+
-	-	-	1	-
-	+	-	2	-
-	+	-	1	-
+	-	-	1	-
-	-	+	2	-
-	+	+	1	-

Eső: -=nem esik az eső; +=esik az eső
Vásár: -=nincs vásár; +=van vásár
Felújítás: -=nincsenek felújítási munkálatok; +=vannak felújítási munkálatok
Időpont: 1=reggel; 2=napközben; 3=este
Késik: -=nem késik a busz; +=késik a busz

K=-: $P(K=-)*P(E=+|\,K=-)*P(V=+|\,K=-)*P(F=-|\,K=-)*P(I=2|\,K=-)=$

$$=\frac{6}{14}*\frac{1}{6}*\frac{3}{6}*\frac{4}{6}*\frac{2}{6}=\frac{1}{126}$$

K=+: $P(K=+)*P(E=+|\,K=+)*P(V=+|\,K=+)*P(F=-|\,K=+)*P(I=2|\,K=+)=$

$$=\frac{8}{14}*\frac{5}{8}*\frac{6}{8}*\frac{3}{8}*\frac{3}{8}=\frac{135}{3584}$$

Mivel $\frac{1}{126}<\frac{1}{100}=\frac{135}{13500}<\frac{135}{3584}$, ezért a táblázatban szereplő adatokon tanított naiv-Bayes modell alapján az a legvalószínűbb, hogy késni fog a buszunk, ha az adott időpontban esik az eső, van vásár a városban, nincsenek felújítási munkálatok az utakon és napközben várunk a buszra.

3. feladat (megoldás nélkül)

A táblázatban felsorolt megfigyelések alapján a naiv-Bayes modellt használva feltehetőleg könnyű lesz-e eladni egy olyan autót, ami japán gyártmányú, 5 évnél fiatalabb, piros és van benne klíma?

Származási ország (O)	5 évnél fiatalabb (F)	Szín (SZ)	Klíma (K)	Könnyű eladni (E)
J	+	P	-	-
N	-	P	-	+
J	+	F	+	-
J	+	S	+	+
N	+	F	+	+
N	-	P	-	-
J	-	F	+	+

Származási ország: J=Japán; N=Németország
5 évnél fiatalabb: -=5 évnél nem fiatalabb; +=5 évnél fiatalabb
Szín: P=piros; F=fekete; S=sárga
Klíma: -=nincs benne klíma; +=van benne klíma
Könnyű eladni: -=nem könnyű eladni; +=könnyű eladni

8. óra - Számítógépes nyelvfeldolgozás, szövegek automatikus osztályozása

Számítógépes nyelvfeldolgozás

A számítógépes nyelvfeldolgozás (vagy számítógépes nyelvészet) témakörébe tartozik minden olyan feladat, mely során számítógépek természetes (emberi) nyelvű szövegekkel dolgoznak. A mesterséges intelligencia egyik alapvető, régóta kutatott területe, mellyel napjainkban is számos kutatás foglalkozik. Rengeteg számítógépes nyelvészeti probléma már részben vagy egészében megoldott, de léteznek nagyon bonyolult feladatok is, melyek megoldása még sokat várat magára.

A számítógépes nyelvfeldolgozás főbb területei:

- Kérdésekre automatikus válasz adása
- Automatikus információkinyerés
- Automatikus fordítás
- Automatikus helyesírás-javítás
- Automatikus összegzés
- Szövegek automatikus osztályozása
- stb.

Szövegek automatikus osztályozása (felügyelt változat)

A szövegek felügyelt automatikus osztályozásának feladata szövegek előre megadott osztályokba való besorolása bemeneti és kimeneti tanító minták alapján, vagyis adott d dokumentum és C osztálycímkék halmaza esetén a d dokumentumhoz legmegfelelőbb C-beli osztálycímke hozzárendelése a $(d_i, f(d_i))$ alakú tanító példák alapján.

Szöveg-osztályozási feladatok a gyakorlatban:

- Automatikus spam szűrés
- Automatikus vélemény-detektálás
- Szövegek témák szerinti automatikus csoportosítása
- Nyelv automatikus detektálása
- stb.

A szövegek osztályozáshoz gyakorlatilag **bármilyen általános osztályozó** algoritmus felhasználható (így pl. a kNN és a naiv-Bayes is), nincs szükség speciálisan szövegek osztályozásához kifejlesztett algoritmusra. Azonban mivel a tanító szövegekhez tartozó tulajdonságok nem adottak explicit módon, ezért a tanítás előtt szükséges a tanító szövegekből a tulajdonságok kinyerésére. Szövegek automatikus osztályozásakor igazából ez a fő megoldandó feladat, ez után lényegében csak a már meglévő osztályozó algoritmusokat kell tesztelni a különféle paraméter-beállításokkal. Szövegek

tulajdonságainak hatékony kinyeréséhez viszont először érdemes néhány **előfeldolgozó lépést** tenni, melyek nagy befolyással lehetnek a tanítás eredményére. Ilyen előfeldolgozó lépések lehetnek például a következők:

- mondatokra bontás
- tokenizálás (szavakra bontás)
- lemmatizálás (szótövesítés)
- azonos típusú, de különböző alakú részek helyettesítése általános tokenekkel (pl. telefonszámok => #TEL#, webcímek => #URL#, email címek => #EMAIL#, dátumok => #DATE#, általános számok => #NUM#)
- stopszavak (olyan nagyon gyakori szavak, melyek szinte minden dokumentumban szerepelnek, és így nem hordoznak értékes információt, pl. a névelők) kiszedése
- feladat-specifikus szótár használata, mely csak a feladat szempontjából fontos szavakat tartalmazza (a szótárban nem szereplő szavak a tanítás során nincsenek figyelembe véve)
- kisbetűsítés

Ez nem azt jelenti, hogy minden feladatban minden előfeldolgozó lépést fel kell használni, ezek a lépések opcionálisak, és a probléma függvényében kell kiválasztani a megfelelőket. Például attól függ, hogy érdemes-e kisbetűsíteni a teljes szöveget, hogy a feladat szempontjából hordoznak-e a kisbetűs és nagybetűs szavak közti különbségek értékes információt.

Az előfeldolgozás után jöhet a **tulajdonságok kinyerése**. Egy szöveget általában a benne szereplő szavakkal, szókapcsolatokkal lehet jellemezni. Ezért a tulajdonságok kinyeréséhez leggyakrabban egy unigram, bigram vagy n-gram modellt használnak, vagyis a szövegekben található szavak, szó-párok vagy szó n-esek, gyakoriságukkal együtt vagy gyakoriságuk nélkül (binárisan), adják a szövegekhez tartozó tulajdonságokat.

A k-legközelebbi-szomszéd (kNN) modell használata szövegek osztályozására

Az algoritmus:

1. Keressük meg a d_i dokumentum k db legközelebbi szomszédját valamilyen szövegekre alkalmazható távolságmérték alapján, pl.:

$$dist(d_i, d_j) = \frac{1}{közösenElőfordulóSzavakSzáma(d_i, d_j)}$$

   ahol $közösenElőfordulóSzavakSzáma(d_i, d_j)$ azt adja meg, hogy hány olyan szóalak létezik, mely a d_i és a d_j dokumentumban is szerepel (gyakoriságtól függetlenül).

2. A $h(d_i)$ értékét határozzuk meg a k db legközelebbi szomszéd f értéke alapján, ami pl. lehet a szomszédok f(d) értékeinek többségi szavazata:

$$h(d_i) = \text{TSZ}_{j=1}^{k}\left(f(d_j)\right)$$

A naiv-Bayes modell használata szövegek osztályozására

A naiv-Bayes osztályozás során abba a $c^* \in C$ osztályba soroljuk a $w_1, \ldots, w_n$ jellemzőkkel leírt (unigram modell esetén a $w_1, \ldots, w_n$ szavakból álló) szöveget, melyre a $P(c \mid w_1, \ldots, w_n)$ valószínűség a legnagyobb, azaz

$$\mathrm{c}^* = \underset{\mathrm{c}\in\mathrm{C}}{\operatorname{argmax}}\ P(c \mid w_1, \ldots, w_n) = \underset{\mathrm{c}\in\mathrm{C}}{\operatorname{argmax}}\, P(c) \prod_{i=1}^{n} P(w_i \mid c)$$

A $P(c)$ illetve a $P(w_i \mid c)$ valószínűségek a megfigyelési adatokból általában könnyen becsülhetők (**empirikus valószínűség**):

$$P(c) \approx \frac{n_c}{N}$$

$$P(w_i \mid c) \approx \frac{n_{c,w_i}}{\sum_{w\in V} n_{c,w}}$$

ahol n_c azoknak a tanító szövegeknek a száma, aminek az osztálya c, N a tanító szövegek száma, V a használt szótár (ha nincs explicit megadva, akkor a tanító halmazban szereplő szóalakok alkotják), n_{c,w_i} pedig a w_i tulajdonság (unigram modell esetén szó) előfordulási gyakorisága a c osztályú szövegekben (egy szövegben egy tulajdonság többször is szerepelhet). Így $\sum_{w\in V} n_{c,w}$ igazából az adja meg, hogy a c osztályú dokumentumok együttesen hány V-ben szereplő szóból állnak.

A túl kevés minta miatt azonban a $P(x_i \mid c)$ valószínűség empirikus becslése gyakran 0 értéket adna. Ilyen esetben az adott osztályhoz tartozó $P(c)\prod_{i=1}^{n} P(w_i \mid c)$ érték 0 lenne, függetlenül a többi valószínűségtől. ezért a gyakorlatban érdemes valamilyen simítást használni. A leggyakrabban a **Laplace-simítást** szokták alkalmazni:

$$P(w_i \mid c) \approx \frac{n_{c,w_i} + 1}{(\sum_{w\in V} n_{c,w}) + |V|}$$

Amennyiben azzal a lehetőséggel is számolunk, hogy a még nem látott tesztdokumentumaink a tanító halmazban nem szereplő szavakat is tartalmaznak, akkor egészítsük ki a szótárunkat egy úgynevezett "ismeretlen szó"-val, és pl. cseréljük le az összes ismeretlen szót a tesztdokumentumainkban a #UNKNOWN# tokenre. Ilyen esetben a képletünk emiatt a plusz szó miatt a következő alakra változik:

$$P(w_i \mid c) \approx \frac{n_{c,w_i} + 1}{\left(\sum_{w\in V} n_{c,w}\right) + |V| + 1}$$

1. feladat

A probléma során azt vizsgáljuk, hogy termékekről adott szöveges értékelések pozitív vagy negatív véleménnyel bírnak a vizsgált termékről. A feladat "Good? Bad! Very Bad!" szöveges értékelésről eldönteni, hogy pozitív vagy pedig negatív, az alábbi táblázatban felsorolt tanító példák alapján:

ID	Szöveg	Osztály
d_1	GOOD	+
d_2	Very good.	+
d_3	BAD!!!	-
d_4	Very Bad.	-
d_5	Very, VERY bad.	-

Az első lépés a szöveg előfeldolgozása, amit mindig az aktuális probléma tulajdonságaitól függően kell elvégezni. Jelen esetben például érdemes a szöveget csupa kisbetűssé alakítani, az írásjeleket törölni, és a szöveget tokenizálni (szavakra bontani). Így a következő előfeldolgozott tanító példákat illetve előfeldolgozott teszt példát kapjuk (a tokeneket ;-vel választjuk el):

ID	Előfelfolgozott szöveg	Osztály
d_1	good	+
d_2	very; good	+
d_3	bad	-
d_4	very; bad	-
d_5	very; very; bad	-
d_6	good; bad; very; bad	?

A tulajdonságok kinyeréséhez ezután használjunk unigram modellt, majd a teszt példa osztályozásához használjunk valamilyen általános osztályozót.

Megoldás kNN osztályozóval

Távolságmetrikának használjuk a $dist(d_i, d_j) = \frac{1}{kölcsönösenElőfordulóSzavakSzáma(d_i,d_j)}$ metrikát.

$dist(d_6, d_1) = \frac{1}{1}$

$dist(d_6, d_2) = \frac{1}{2}$

$dist(d_6, d_3) = \frac{1}{1}$

$dist(d_6, d_4) = \frac{1}{2}$

$dist(d_6, d_5) = \frac{1}{2}$

k = 3 => $h(d_6) = \text{TSZ}(f(d_2), f(d_4), f(d_5)) = TSZ(+, -, -) = -$

k = 5 => $h(d_6) = \text{TSZ}(f(d_1), f(d_2), f(d_3), f(d_4), f(d_5)) = TSZ(+, +, -, -, -) = -$

Tehát 3 és 5 szomszéd figyelembe vétele esetén is a kNN osztályozó negatív értékelésnek ítéli meg a d_6 teszt szöveget.

Megoldás naiv-Bayes osztályozóval

A 0 valószínűségek elkerüléséhez használjunk valamilyen simítást a $P(x_i \mid c)$ valószínűségek számításánál. Mivel a teszt szövegben csak a tanító szövegekben is szereplő szavak vannak, ezért elegendő egyszerű Laplace-simítást alkalmazni, az ismeretlen szavakkal nem kell foglalkozzunk. Tehát:

$$P(w_i \mid c) \approx \frac{n_{c,w_i} + 1}{(\sum_{w \in V} n_{c,w}) + |V|}$$

C=+: $P(C = +) * P(good \mid C = +) * P(bad \mid C = +) * P(very \mid C = +) * P(bad \mid C = +) =$

$$= \frac{2}{5} * \frac{2+1}{3+3} * \frac{0+1}{3+3} * \frac{1+1}{3+3} * \frac{0+1}{3+3} = \frac{2}{5} * \frac{1}{2} * \frac{1}{6} * \frac{1}{3} * \frac{1}{6} = \frac{1}{540} \approx 0{,}0019$$

C=-: $P(C = -) * P(good \mid C = -) * P(bad \mid C = -) * P(very \mid C = -) * P(bad \mid C = -) =$

$$= \frac{3}{5} * \frac{0+1}{6+3} * \frac{3+1}{6+3} * \frac{3+1}{6+3} * \frac{3+1}{6+3} = \frac{3}{5} * \frac{1}{9} * \frac{4}{9} * \frac{4}{9} * \frac{4}{9} = \frac{64}{10935} \approx 0{,}0059$$

Mivel $0{,}0059 > 0{,}0019$, ezért a naiv-Bayes osztályozó szintén negatív értékelésnek ítéli meg a d_6 teszt szöveget.

2. feladat

A feladat egy egyszerű spam szűrő tanítása, ami egy még nem látott e-mailről el tudja azt dönteni, hogy spam-e vagy sem. A teszt e-mail szövege: "You have won the very huge 1 MILLION DOLLAR prize!!!!!", melyet az alábbi táblázatban felsorolt tanító példák alapján kellene osztályozni:

ID	E-mail	Osztály
d_1	Click HERE to win the PRIZE!	+
d_2	You inherited 10 million dollars!!!	+
d_3	Very Huge PRIZE!	+
d_4	Meeting you at 5 pm?	-
d_5	CALL me back!	-
d_6	Should we order a huge PIZZA?	-
d_7	The lottery prize is 3 MILLION dollars this week...	-

Az első lépés a szöveg előfeldolgozása, amit mindig az aktuális probléma tulajdonságaitól függően kell elvégezni. Jelen esetben például érdemes az e-maileket csupa kisbetűssé alakítani, az írásjeleket és a névelőket törölni, a számokat #NUM#-re cserélni, a szavakat lemmatizálni (szótövesíteni) és a szöveget tokenizálni (szavakra bontani). Mivel a teszt e-mail tartalmaz olyan szót is, ami nem szerepel a tanító e-mailekben, ezért az ismeretlen szót helyettesítsük az #UNKNOWN# tokennel. Így a következő előfeldolgozott tanító példákat illetve előfeldolgozott teszt példát kapjuk (a tokeneket ;-vel válaszjuk el):

ID	Előfeldolgozott e-mail	Osztály
d_1	click; here; to; win; prize	+
d_2	you; inherit; #NUM#; million; dollar	+
d_3	very; huge; prize	+
d_4	meet; you; at; #NUM#; pm	-
d_5	call; me; back	-
d_6	should; we; order; huge; pizza	-
d_7	lottery; prize; is; #NUM#; million; dollar; this; week	-
d_8	you; #UNKNOWN#; win; very; huge; #NUM#; million; dollar; prize	?

A tulajdonságok kinyeréséhez ezután használjunk unigram modellt, majd a teszt példa osztályozásához használjunk valamilyen általános osztályozót.

Megoldás kNN osztályozóval

Távolságmetrikának használjuk a $dist(d_i, d_j) = \frac{1}{kölcsönösenElőfordulóSzavakSzáma(d_i,d_j)}$ metrikát.

$$dist(d_8, d_1) = \frac{1}{2}$$

$$dist(d_8, d_2) = \frac{1}{4}$$

$$dist(d_8, d_3) = \frac{1}{3}$$

$$dist(d_8, d_4) = \frac{1}{2}$$

$$dist(d_8, d_5) = \frac{1}{0}$$

$$dist(d_8, d_6) = \frac{1}{1}$$

$$dist(d_8, d_7) = \frac{1}{4}$$

k = 3 => $h(d_8) = \text{TSZ}\big(f(d_2), f(d_3), f(d_7)\big) = TSZ(+, +, -) = +$

k = 5 => $h(d_8) = \text{TSZ}\big(f(d_1), f(d_2), f(d_3), f(d_4), f(d_7)\big) = TSZ(+, +, +, -, -) = +$

Tehát 3 és 5 szomszéd figyelembe vétele esetén is a kNN osztályozó spamnek ítéli meg a d_8 teszt e-mailt.

Megoldás naiv-Bayes osztályozóval

A 0 valószínűségek elkerüléséhez használjunk valamilyen simítást a $P(x_i \mid c)$ valószínűségek számításánál. Mivel a teszt szövegben szerepel a tanító szövegekben nem szereplő szó is, ezért használjunk Laplace-simítást az ismeretlen szavak figyelembe vételével együtt. Tehát:

$$P(w_i \mid c) \approx \frac{n_{c,w_i} + 1}{\left(\sum_{w \in V} n_{c,w}\right) + |V| + 1}$$

C=+: $P(C = +) * P(you|\ C = +) * P(\#UNKNOWN\#|\ C = +) * P(win|\ C = +) *$

$* P(very|\ C = +) * P(huge|\ C = +) * P(\#NUM\#|\ C = +) *$

$* P(\text{million}|\ C = +) * P(\text{dollar}|\ C = +) * P(\text{prize}|\ C = +) =$

$$= \frac{3}{7} * \frac{1+1}{13+26+1} * \frac{0+1}{13+26+1} * \frac{1+1}{13+26+1} * \frac{1+1}{13+26+1} * \frac{1+1}{13+26+1} * \frac{1+1}{13+26+1} *$$

$$* \frac{1+1}{13+26+1} * \frac{1+1}{13+26+1} * \frac{2+1}{13+26+1} = \frac{3}{7} * \frac{2}{40} * \frac{1}{40} * \frac{2}{40} * \frac{2}{40} * \frac{2}{40} * \frac{2}{40} * \frac{2}{40} * \frac{2}{40} *$$

$$* \frac{3}{40} = \frac{1152}{7 * 40^9} \approx 6{,}2779 * 10^{-13}$$

C=-: $P(C = -) * P(you|\ C = -) * P(\#UNKNOWN\#|\ C = -) * P(win|\ C = -) *$

$* P(very|\ C = -) * P(huge|\ C = -) * P(\#NUM\#|\ C = -) *$

$* P(\text{million}|\ C = -) * P(\text{dollar}|\ C = -) * P(\text{prize}|\ C = -) =$

$$= \frac{4}{7} * \frac{1+1}{21+26+1} * \frac{0+1}{21+26+1} * \frac{0+1}{21+26+1} * \frac{0+1}{21+26+1} * \frac{1+1}{21+26+1} * \frac{2+1}{21+26+1} *$$

$$* \frac{1+1}{21+26+1} * \frac{1+1}{21+26+1} * \frac{1+1}{21+26+1} = \frac{4}{7} * \frac{2}{48} * \frac{1}{48} * \frac{1}{48} * \frac{1}{48} * \frac{2}{48} * \frac{3}{48} * \frac{2}{48} * \frac{2}{48} *$$

$$* \frac{2}{48} = \frac{384}{7 * 48^9} \approx 4{,}0557 * 10^{-14}$$

Mivel $6{,}2779 * 10^{-13} > 4{,}0557 * 10^{-14}$, ezért a naiv-Bayes osztályozó szintén spamnek ítéli meg a d_8 teszt e-mailt.

9. óra - Döntési fák

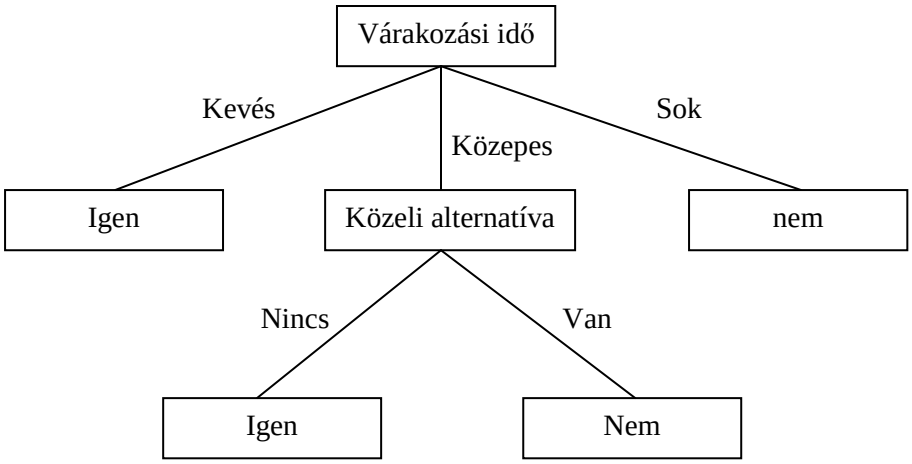
Döntési fák

A döntési fák tanulása az egyik legegyszerűbb, mégis az egyik leggyakrabban alkalmazott felügyelt tanulási módszer. Az eddig tárgyalt algoritmusokhoz hasonlóan itt is bemeneti és kimeneti minták adottak, és ebből kell megtanulni a kimeneti értékeket képző függvényt. A bemeneteket attribútumokkal írjuk le, ezek és a kimenetek is lehetnek diszkrétek vagy folytonosak, de a gyakorlaton csak diszkrét esetekkel foglalkozunk.

A döntési fa egy **tesztsorozat elvégzése** után jut el a döntéshez, ahol a fának:

- a belső csúcsai a bemeneti attribútumokhoz tartozó teszteknek,
- a csomópontokból kilépő ágak a tesztek lehetséges kimeneteleinek felelnek meg,
- a levélcsomópontok adják meg azt azokat az értékeket, amivel a döntési fa visszatér, ha az adott csomópontot elértük.

Egy példa döntési fa annak eldöntésére, hogy beülünk-e egy vendéglőbe annak függvényében, hogy mennyi a várakozási idő illetve hogy van-e a közelben másik olyan vendéglő, ahol szintén szívesen ennénk:



A döntési fák kifejezőereje az **ítéletkalkulus kifejezőerejével** egyezik meg. Sajnos ez azt jelenti, hogy egyszerűbb problémákat jól tudnak reprezentálni, de bonyolultakat már nem hatékonyan (pl. már a többségi szavazás megvalósítása is nagyon nem-hatékony).

Cél: egy olyan döntési fa készítése, mely **minimális mélységű**, de konzisztens az adatokkal.

Megjegyzések: Nem feltétlen kell az összes attribútumot felhasználni ahhoz, hogy konzisztens fát kapjunk, és ezzel elkerülhető a túlillesztés is. Az viszont előfordulhat, hogy egy attribútumot a fa több különböző ágában is tesztelünk.

Döntési fák tanulása az ID3 algoritmussal

Az ID3(S, F) algoritmus lépései (S a példák, F az attribútumok halmaza, az algoritmus kezdetben az összes példára és a teljes attribútum-halmazra hívódik meg):

1. Ha minden megmaradt példa egy adott osztályhoz tartozik, akkor adjuk vissza az adott osztály címkéjét tartalmazó levélcsomópontot.
2. Ha nem maradt meg példánk, akkor adjuk vissza a szülő csomópontban szereplő példák többségi szavazatát tartalmazó levélcsomópontot.
3. Ha maradt még pozitív és negatív példánk is, de már minden attribútumot felhasználtunk a teszteléshez, akkor térjünk vissza a megmaradt példák többségi szavazatát tartalmazó levélcsomóponttal. Ekkor az adatok zajosak (nem konzisztensek), és így a döntési fánk sem lesz konzisztens.
4. Egyéb esetben válasszuk ki a legjobban szeparáló, még fel nem használt attribútumot a megmaradt példáink szétválasztására. A kiválasztott attribútumból létrehozunk egy belső csomópontot, az ebből kilépő ágak lesznek az attribútum lehetséges értékei. A kiválasztott attribútum gyerekei pedig azok a csomópontok lesznek, amiket a létrejött példahalmazokra az algoritmus rekurzív hívásával kapunk.

   Rekurzív hívás: minden v ∈ R_{LSZA} értékre az ID3(S_v, F\{LSZA}) rekurzív hívás adja meg az adott ághoz tartozó gyerek csomópontot, ahol LSZA a legjobban szeparáló attribútum, R_{LSZA} az LSZA attribútum értékkészlete, S_v pedig az a részhalmaza S-nek, melynek minden elemére LSZA = v.

A legjobban szeparáló attribútum

Az az attribútum lesz a legjobban szeparáló, amelynek a tesztjével a legnagyobb **információnyereséget** tudjuk elérni. Egy adott attribútum tesztjével járó információnyereség:

$$Gain(S, A) = Entropy(S) - \sum_{v \in R_A} \frac{|S_v|}{|S|} Entropy(S_v)$$

ahol R_A az A attribútum értékkészlete, S_v az a részhalmaza S-nek, melynek minden elemére A = v, és Entropy(S) az S halmaz entrópiája. Az entrópia számításának számos módja létezik, a **Shannon-féle entrópia** például a következőképpen számolható ki egy tetszőleges S halmazra:

$$Entropy(S) = -\sum_{i \in R_S} p_i log(p_i)$$

ahol R_S az S halmaz értékkészlete (osztálycímkék halmaza), p_i pedig az i osztály előfordulási valószínűsége. Ez a p_i valószínűség a megfigyelési adatokból általában könnyen becsülhető (empirikus valószínűség):

$$p_i \approx \frac{n_i}{N}$$

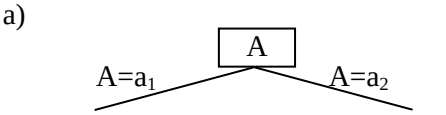
ahol n_i azoknak az S-beli példáknak a száma, aminek az osztálya i, N pedig az S halmaz számossága.

1. feladat

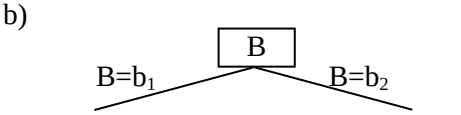
Az ID3 futása közben a konstruálás alatt álló fa egyik csúcsánál a lenti ábrán látható döntési helyzet adódik - azaz a csúcs vagy A címkét kap, és ezáltal az a) eset áll elő, vagy B címkét kap, és ezáltal a b) helyzet áll elő. A Shannon-féle entrópia helyett a Gini-féle $E(S) = 4p_+p_-$ indexet használva melyik attribútumot választja az ID3, és miért? (S a csúcshoz tartozó példák halmaza; $S_{F=f}$ azon S-beli példák halmaza, amelyeknél az F attribútum f értéket vesz fel; p_+ a pozitív példák, p_- a negatív példák előfordulási valószínűsége S-ben; ℓ első komponense mindig a paraméterében lévő példahalmaz pozitív, a második pedig a negatív elemeinek száma.)

ℓ(S)= (10, 20)

a)



b)



$$Entropy(S) = 4 * \frac{10}{30} * \frac{20}{30} = \frac{8}{9}$$

$$Gain(S,A) = \frac{8}{9} - \left(\frac{25}{30} * 4 * \frac{10}{25} * \frac{15}{25} + \frac{5}{30} * 4 * \frac{0}{5} * \frac{5}{5}\right) = \frac{8}{9} - \left(\frac{4}{5} + 0\right) = \frac{8}{9} - \frac{4}{5} \approx 0{,}0889$$

$$Gain(S,B) = \frac{8}{9} - \left(\frac{12}{30} * 4 * \frac{4}{12} * \frac{8}{12} + \frac{18}{30} * 4 * \frac{6}{18} * \frac{12}{18}\right) = \frac{8}{9} - \left(\frac{16}{45} + \frac{8}{15}\right) = \frac{8}{9} - \frac{40}{45} = 0$$

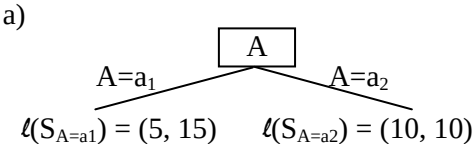
Mivel az A attribútum választásával nagyobb az információnyereség, mint a B attribútum választásával, ezért az ID3 az A attribútumot választja.

2. feladat

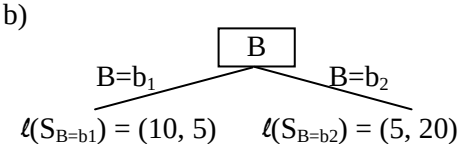
Az ID3 futása közben a konstruálás alatt álló fa egyik csúcsánál a lenti ábrán látható döntési helyzet adódik - azaz a csúcs vagy A címkét kap, és ezáltal az a) eset áll elő, vagy B címkét kap, és ezáltal a b) helyzet áll elő. A Shannon-féle entrópia helyett a Gini-féle $E(S) = 4p_+p_-$ indexet használva melyik attribútumot választja az ID3, és miért? (S a csúcshoz tartozó példák halmaza; $S_{F=f}$ azon S-beli példák halmaza, amelyeknél az F attribútum f értéket vesz fel; p_+ a pozitív példák, p_- a negatív példák előfordulási valószínűsége S-ben; ℓ első komponense mindig a paraméterében lévő példahalmaz pozitív, a második pedig a negatív elemeinek száma.)

ℓ(S)= (15, 25)

a)



b)



$$Entropy(S) = 4 * \frac{15}{40} * \frac{25}{40} = \frac{15}{16}$$

$$Gain(S,A) = \frac{15}{16} - \left(\frac{20}{40} * 4 * \frac{5}{20} * \frac{15}{20} + \frac{20}{40} * 4 * \frac{10}{20} * \frac{10}{20}\right) = \frac{15}{16} - \left(\frac{3}{8} + \frac{1}{2}\right) = \frac{15}{16} - \frac{7}{8} \approx 0{,}0625$$

$$Gain(S,B) = \frac{15}{16} - \left(\frac{15}{40} * 4 * \frac{10}{15} * \frac{5}{15} + \frac{25}{40} * 4 * \frac{5}{25} * \frac{20}{25}\right) = \frac{15}{16} - \left(\frac{1}{3} + \frac{2}{5}\right) = \frac{15}{16} - \frac{11}{15} \approx 0{,}2042$$

Mivel a B attribútum választásával nagyobb az információnyereség, mint az A attribútum választásával, ezért az ID3 a B attribútumot választja.

3. feladat (megoldás nélkül)

A táblázatban felsorolt megfigyelések alapján az ID3 algoritmust futtatva készítsünk egy olyan döntési fát, mely segítségével még nem látott megfigyelési adatokra is eldönthető, hogy a megfigyelt személy a megfigyelt időjárási körülmények között fog-e teniszt játszani vagy sem. A Shannon-féle entrópia helyett használjuk a Gini-féle $E(S) = 4p_+p_-$ indexet (p_+ a pozitív példák, p_- a negatív példák előfordulási valószínűsége S-ben).

Outlook	Temperature	Humidity	Wind	Play tennis
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Overcast	Hot	High	Weak	Yes
Rain	Mild	High	Weak	Yes
Rain	Cool	Normal	Weak	Yes
Rain	Cool	Normal	Strong	No
Overcast	Cool	Normal	Strong	Yes
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Rain	Mild	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes
Overcast	Mild	High	Strong	Yes
Overcast	Hot	Normal	Weak	Yes
Rain	Mild	High	Strong	No

10. óra - Lokális keresés, optimalizálás

Globális optimalizálás

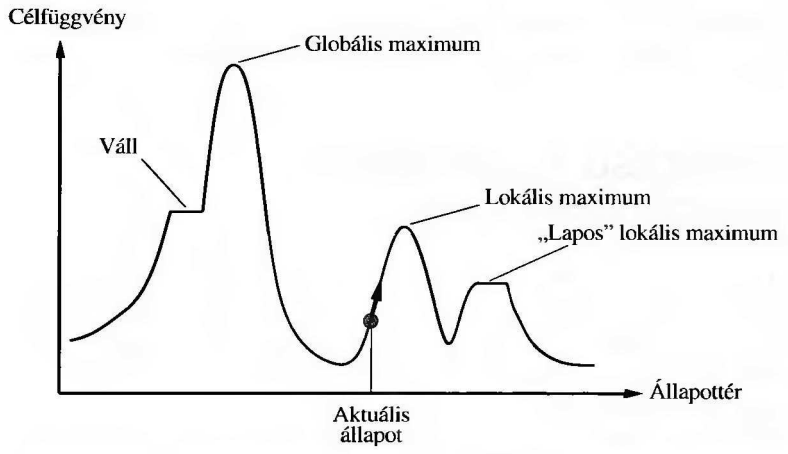
Az **informálatlan és informált keresőalgoritmusoknál** az állapottérben való kereséskor egy optimális célállapotot és a hozzá vezető utat kerestük, és a költség az út függvénye volt. Igazából maga a célhoz vezető út volt a probléma megoldása.

A **globális optimalizálási problémáknál** egy másfajta keresést alkalmazunk: a költség az állapot függvénye (az út ekkor lényegtelen). Itt maga a célállapot a megoldás, a célhoz vezető út lényegtelen.

Az alkalmazott **modell** a következő:

- **lehetséges állapotok halmaza** (state space)
- **(kezdőállapot(ok) lehet(nek)**, de ez igazából nem része a probléma definíciójának mert az állapotok értékelése nem függ a kezdőállapottól)
- **állapotátmenet függvény**: minden állapothoz hozzárendel egy (operátor, követő-állapot) típusú rendezett párokból álló halmazt
- **célfüggvény** (objective function): állapotok f értékelő függvénye, amely minden lehetséges állapothoz valós értéket rendel
- **célállapotok halmaza** (lehetséges állapotok részhalmaza): a **globális maximumhelyek** (néha globális minimumhelyek) halmaza: $\{x \mid f(x) = \max_{y \text{ egy állapot}} f(y)\}$.

Az állapotok és operátorok által definiált gráfot itt **optimalizálási tájnak** (landscape) hívjuk, amelyben hegycsúcsok, völgyek, hegygerincek, vállak, fennsíkok stb. vannak. Vigyázat: egy sokdimenziós táj nem intuitív, sokdimenziós tájban a legmeredekebb emelkedő irányát sem lehet intuitív meghatározni!



Az optimalizáló algoritmusok általában teljes állapotleírást használnak, a teljes állapotok között lépegetnek az operátorok segítségével.

Példa: **8 királynő probléma**:

- **Állapotok**: számnyolcasok, ahol az i. szám azt mutatja, hogy az i. sorban hányadik oszlopban helyezkedik el a királynő
- **Kezdőállapot**: nem kell megadni, tetszőleges állapot lehet
- **Állapotátmenet**: egy királynő soron belüli elmozgatása
- **Célfüggvény**: egymást ütő királynő-párok száma
- **Célállapotok halmaza**: olyan állapotok halmaza, ahol a célfüggvény értéke 0 (ezért ez egy minimalizálási probléma)

Az informálatlan és informált keresőalgoritmusok ezzel szemben gyakran egy kezdeti állapotot alkalmaztak, és ebből építették fel a megoldást. De azért az informálatlan és informált keresőalgoritmusoknál is lehet teljes állapotleírás, pl. palacsinták sorba rakása, 8-as kirakó stb.

Egy **teljes** optimalizáló algoritmus mindig talál megoldást, ha az egyáltalán létezik. Mivel ezeknél a problémáknál a globális optimumok a megoldások, így igazából nincs értelme optimalitást nézni.

Egy optimalizáló algoritmusnak gyakran van analógja a korábbi fa- ill. gráfkeresés területén.

Lokális keresés

Az optimalizálási probléma egy mohó megközelítése. A lokális keresőalgoritmusok általában csak egy aktuális állapotot tárolnak, az állapotba vezető útvonalakat nem tárolják és a követő állapot az aktuális állapot szomszédjai közül kerül ki.

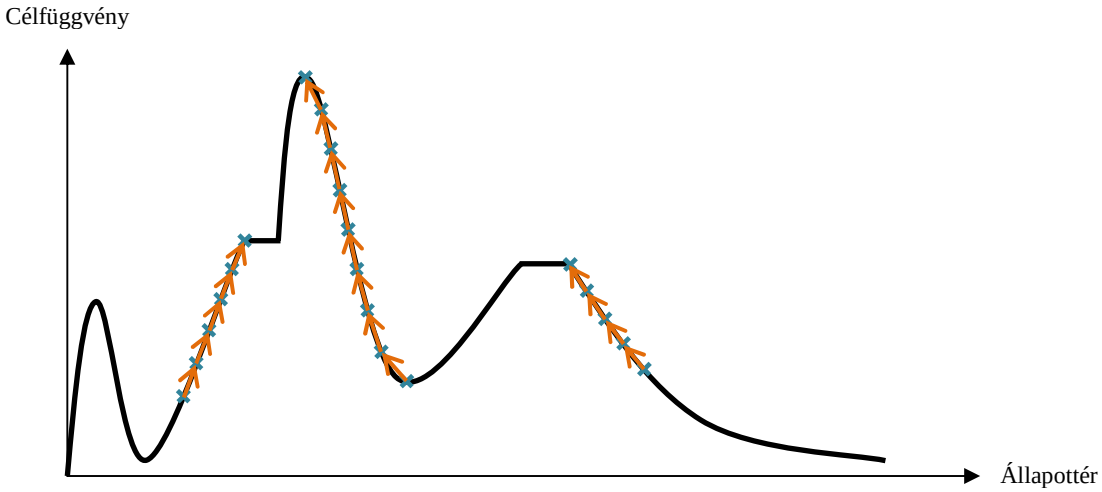
Hegymászó keresés

A keresés egyszerűen csak egy ciklus, ami mindig **javuló értékek felé** - azaz felfelé - lép. A legmeredekebb emelkedő (steepest ascent) változat esetében mindig a legmeredekebb emelkedő irányába lép el (folytonos állapottér esetében a gradiens meghatározása szükséges ehhez, diszkrét esetben pedig az algoritmus a legjobb szomszédba lép). Az algoritmus megáll, amikor felér a csúcsra, ahol nincsenek már magasabb értékű szomszédjai.

Hatékonyság:

- nem teljes: gyakran megakad a lokális maximumok, fennsíkok és vállak miatt
  pl. a 8 királynő probléma esetén 14% eséllyel jár sikerrel
- mégis gyakran alkalmazzák valamilyen javított változatát, mert általában gyorsan halad a megoldás felé, és különféle módszerekkel sokat lehet javítani az algoritmuson, egyes megoldásokkal pl. 1-hez tartó valószínűséggel teljes

A hegymászó keresés grafikus szemléltetése:



Javítási lehetőségek:

- **Oldallépések megengedése**: Vállakat ki lehet küszöbölni vele. 8 királynő problémán már 94% siker (tehát ennél a problémánál sok a váll).
- **Sztochasztikus hegymászó**: Véletlen szomszéd azok közül, amelyek jobbak (nem feltétlenül a legjobb). Lassabb, de ritkábban akad el.
- **Első választásos hegymászó**: Vegyük az első szomszédot, ami jobb. Ez a sztochasztikus hegymászó egy implementációja, és akkor hatékonyabb, ha nagyon sok szomszéd van.
- **Véletlen újraindított hegymászó**: Ha nem sikeres, akkor indítsuk újra véletlen kezdőpontból. Ez nagyon jó algoritmus, pl. 3 millió királynő probléma megoldása egy perc alatt! De a keresési tér struktúrájától nagyban függ. 1-hez tartó valószínűséggel teljes, míg az eddigiek nem voltak teljesek. Minél többször indítjuk újra véletlenszerűen, annál nagyobb valószínűséggel találja meg a globális optimumot.

Szimulált lehűtés

Alapötlet: A hegymászó keresés nem teljes, de hatékonyan keresi a megoldást. A véletlen vándorlás 1-hez tartó valószínűséggel teljes, de nagyon nem-hatékony. Ennek a kettőnek az ötvözése úgy, hogy a teljesség és a hatékonyság is megmaradjon.

Az algoritmus:

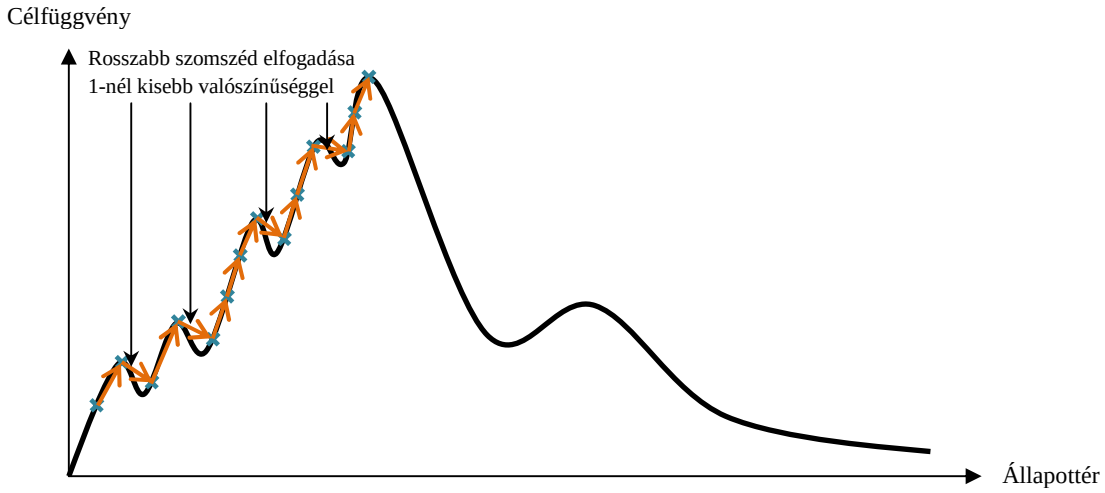
1. Kiválasztja az aktuális állapot egy véletlen szomszédját.
2. Ha a szomszéd célfüggvény-értéke magasabb az aktuális állapoténál, akkor a követő állapot ez a szomszéd lesz.
3. Ha a szomszéd célfüggvény-értéke nem magasabb az aktuális állapoténál, akkor követőnek ez a szomszéd valamilyen 1-nél kisebb valószínűséggel lesz választva (egyébként a követő marad az aktuális állapot), ahol ez a valószínűség exponenciálisan csökken a szomszéd célfüggvény-

értékének rosszságával, továbbá csökken a T hőmérséklet csökkenésével is. Ez a T hőmérséklet időben csökken, tehát egyre kevésbé lesz valószínű a rossz irányba elmozdulás. Ez a lehűtés rész.

4. goto 1

1-hez tartó valószínűséggel teljes. Minél hosszabb a hűtési folyamat, annál nagyobb valószínűséggel találja meg a globális optimumot.

A szimulált lehűtés grafikus szemléltetése:



Populáció alapú lokális keresés

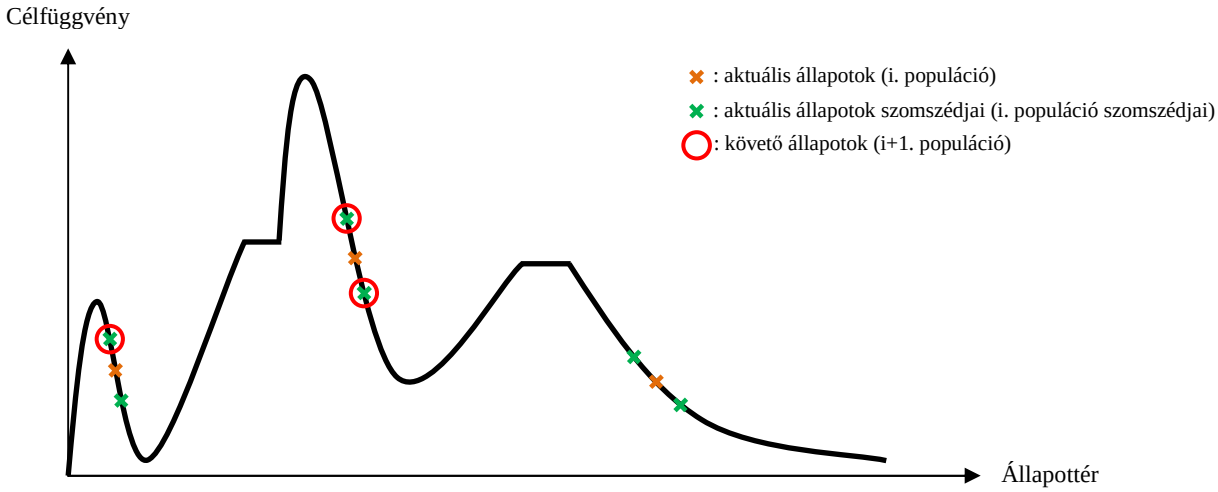
A populáció alapú lokális keresők egy állapot helyett egyszerre több állapottal dolgoznak.

Nyaláb keresés

Hasonlít a hegymászó keresésre, de k db véletlen generált állapottal indul (**populáció**). Minden ciklusban veszi ezen állapotok összes követőjét, és ezek közül a k db legjobbat választja ki követő populációnak.

Nem ugyanaz, mint k db független hegymászó: jobban fókuszál, de gyorsabban be is ragad.

A nyaláb keresés egy ciklusának grafikus szemléltetése:



Vannak ugyanúgy javításai, mint a hegymászónak, pl. sztochasztikus, stb.

Evolúciós algoritmusok (EA)

A nyaláb keresés a hegymászó általánosítása volt populációval (k db állapot) és szelekcióval.

Az evolúciós algoritmusok a nyaláb keresés általánosításai **szexuális operátorokkal**.

Genetikus algoritmus (GA)

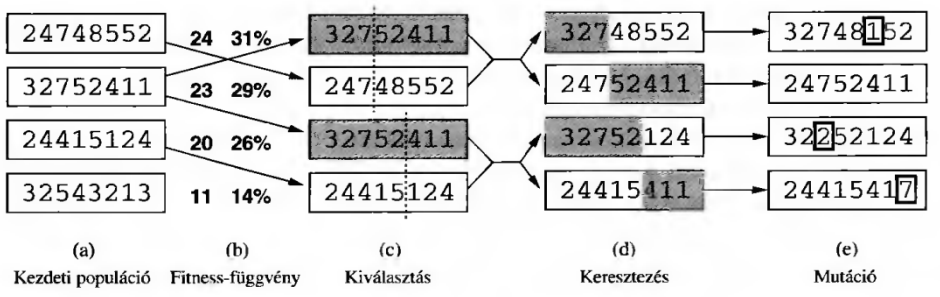
Ez az algoritmus eltér a könyv algoritmusától, itt az előadás szerinti algoritmust vesszük.

Az állapotok véges betűkészletű sztringek (gyakran bináris sztringek).

Az algoritmus:

1. k db egyed (állapot) véletlen generálása (ez a kezdő populáció)
2. k/2 db szülőpár (tehát összesen k db egyed) kiválasztása az aktuális populációból valamilyen módszer alapján (általában egy egyed több szülőpárba is kiválasztható). Egy gyakori módszer a fitnesz-értékkel arányos valószínűséggel történő kiválasztás (a fitnesz-függvény a célfüggvény elnevezése a genetikus algoritmusoknál, mely minden egyedhez egy jósági értéket rendel).
3. szülők rekombinációjával k db új egyed létrehozása
   rekombináció: véletlen keresztezési pontnál vágás és csere
4. új egyedek mutációja valamilyen kis valószínűséggel
   mutáció: pl. egy betű véletlen értékre történő megváltoztatása
5. k db állapot kiválasztása a régiek és újak uniójából túlélő szelekció segítségével. Ez a k db állapot lesz a következő populáció. A túlélő szelekció lehet a k db legjobb egyed kiválasztása, vagy a fitneszfüggvénnyel arányos valószínűséggel k db egyed választása, vagy k db véletlen párnál minden párból a jobb kiválasztása, stb.
6. goto 2

Genetikus algoritmus grafikus szemléltetése:



11. óra - Bayes-hálók

Bayes-hálókat valószínűségi modellek leírására használhatunk. A valószínűségi modellek egyik legkézenfekvőbb megadási módja a teljes együttes eloszlás megadása lenne. azonban n db logikai változó esetén 2^n db elemi esemény valószínűségét kellene megadni és tárolni. Hogy egyszerűbb dolgunk legyen, érdemes a valószínűségi változók közötti **feltételes függetlenségeket** kihasználni, mert segítségükkel tömöríthetjük a teljes együttes eloszlás reprezentációját. A Bayes-hálók a teljes együttes eloszlás feltételes függetlenségeit ragadják meg egy meghatározott módon, és tömör és intuitív reprezentációt (vagy közelítést) tesznek lehetővé. Nagyon nagy témakör, ezért a gyakorlaton csak kis részével tudunk foglalkozni.

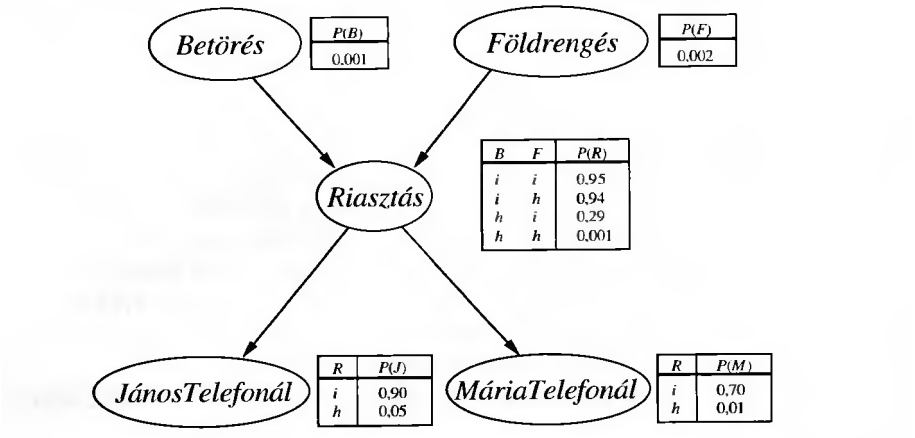
A Bayes-háló egy olyan **irányított, körmentes gráf**, ahol:

1. A háló csomópontjai valószínűségi változók.
2. Az irányított élek csomópontokat kötnek össze. Ha X-ből vezet Y-ba él, akkor X az Y szülője, és X nem független Y-tól. (De nem igaz hogy ha X és Y nem független, akkor van köztük él.)
3. Minden X_i csomóponthoz tartozik egy $P(X_i|\ \text{Szülők}(X_i))$ feltételes valószínűség eloszlás, ami számszerűen megadja a szülők hatását a csomóponti változóra.

Egy Bayes-háló a benne szereplő valószínűségi változók felett a következőképpen definiálja a **teljes együttes eloszlást**:

$$P(X_1 \dots X_n) = \prod_{i=1}^{n} P(X_i|\ \text{Szülők}(X_i))$$

Példa Bayes-háló:



Bayes-hálók konstruálása

Egy adott teljes együttes eloszláshoz nem csak egy Bayes-háló rendelhető, hanem sok különböző. Fontos az, hogy úgy próbáljuk meg a Bayes-hálót megkonstruálni, hogy minden csomóponthoz minél kevesebb másik csomópont kapcsolódjon, mert ez nagyban befolyásolja a Bayes-háló tömörségét (az

élek nem feltétlen ok-okozati relációknak felelnek meg, de annak is megfelelhetnek). Egy olyan Bayes-hálóhoz, amiben minden csúcshoz legfeljebb k db szülő tartozik, összesen legfeljebb $n2^k$ db feltételes valószínűség megadása szükséges. Ha k kicsi, akkor ez sokkal kisebb mint 2^n.

Egzakt következtetés Bayes-hálókban

Egy $Q = q$ lekérdezésre és $E = e$ megfigyelt eseményre:

$$P(Q = q \mid E = e) = \frac{P(Q = q, E = e)}{P(E = e)} = \frac{\sum_y P(Q = q, E = e, Y = y)}{\sum_{q'} \sum_y P(Q = q', E = e, Y = y)}$$

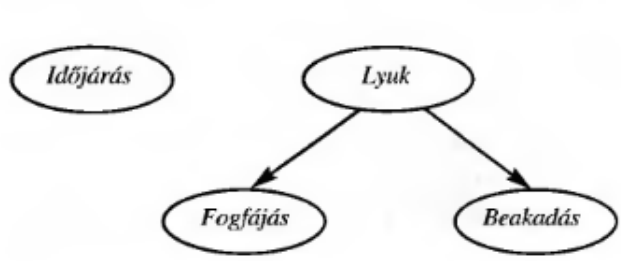
ahol y a meg nem figyelt változók értékeinek összes lehetséges kombinációján, q' pedig a Q célváltozó összes lehetséges értékén fut végig. Ehhez az átalakításhoz először a feltételes valószínűség definícióját, majd a marginalizálási lépést használtuk fel. Az így kapott képlet pedig könnyen kiszámolható a Bayes-háló által megadott feltételes valószínűségekből a teljes valószínűség eloszlás képletével:

$$P(Q = q \mid E = e) = \frac{P(Q = q, E = e)}{P(E = e)} = \frac{\sum_y P(Q = q, E = e, Y = y)}{\sum_{q'} \sum_y P(Q = q', E = e, Y = y)} =$$

$$= \frac{\sum_y \prod_{i=1}^{n} P\big(X_i = x_i \mid szülők(x_i)\big)}{\sum_{q'} \sum_y \prod_{i=1}^{n} P\big(X_i = x_i \mid szülők(x_i)\big)}$$

Az így kapott képletben szereplő valószínűségek már egy az egyben kiolvashatók a Bayes-háló feltételes valószínűségi eloszlásaiból. Tehát nincs más hátra, mint az értékeket behelyettesíteni, és a számolást elvégezni.

Érdemes azonban megjegyezni, hogy egyes esetekben lehetőség van további **egyszerűsítésre**: a szorzatból bármely levélcsomópontot eltávolíthatunk, ami nem célváltozó (Q) vagy bizonyítékváltozó (E), mert azokra összegezni fogunk, és az összegük 1 lesz. Példa:



$$P(F = i \mid LY = h) = \frac{P(F = i, LY = h)}{P(LY = h)} = \frac{\sum_{id} \sum_b P(F = i, LY = h, ID = id, B = b)}{\sum_f \sum_{id} \sum_b P(F = f, LY = h, ID = id, B = b)} =$$

$$= \frac{P(F = i \mid LY = h) P(LY = h) \sum_{id} P(ID = id) \sum_b P(B = b)}{P(LY = h) \sum_f P(F = f \mid LY = h) \sum_{id} P(ID = id) \sum_b P(B = b)} =$$

$$= \frac{P(F = i \mid LY = h) P(LY = h) \sum_{id} P(ID = id)}{P(LY = h) \sum_f P(F = f \mid LY = h) \sum_{id} P(ID = id)} =$$

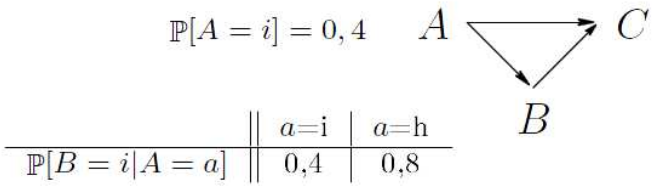
$$= \frac{P(F = i \mid LY = h) P(LY = h)}{P(LY = h) \sum_f P(F = f \mid LY = h)} = \frac{P(F = i \mid LY = h) P(LY = h)}{P(LY = h)}$$

Itt az is látszik, hogy a nevezőt igazából nem is kellett volna a $P(LY = h)$ alakról átalakítani, mert ez a valószínűség már közvetlenül szerepel a Bayes-háló feltételes valószínűségi eloszlásaiban.

Az egzakt következtetés idő-komplexitása Bayes-hálókban $O(n2^n)$, ezért a gyakorlatban gyakran közelítő következtetést alkalmaznak nagy Bayes-hálók esetében. A gyakorlaton mi csak az egzakt esettel foglalkozunk.

1. feladat

A lenti Bayes-hálóval megadott modellben mekkora a $P(A = \text{i} \mid \text{C} = \text{h})$ valószínűség?



	a=i	a=h
$\mathbb{P}[B = i \mid A = a]$	0,4	0,8

a	b	$\mathbb{P}[C = i \mid A = a, B = b]$
i	i	0,5
i	h	0,7
h	i	0,3
h	h	0,9

$$P(A = \text{i} \mid \text{C} = \text{h}) = \frac{P(A = \text{i}, \text{C} = \text{h})}{P(C = h)} = \frac{\sum_b P(A = i, B = b, C = h)}{\sum_a \sum_b P(A = a, B = b, C = h)} =$$

$$= \frac{P(A = i) \sum_b P(B = b \mid A = i) P(C = h \mid A = i, B = b)}{\sum_\text{a} P(A = a) \sum_b P(B = b \mid A = a) P(C = h \mid A = a, B = b)} =$$

$$= \frac{0,4 * (0,4 * 0,5 + 0,6 * 0,3)}{0,4 * (0,4 * 0,5 + 0,6 * 0,3) + 0,6 * (0,8 * 0,7 + 0,2 * 0,1)} =$$

$$= \frac{0,152}{0,152 + 0,348} = 0,304$$

2. feladat

Ugyanezen Bayes-hálóval adott modellben mekkora valószínűséggel teljesül, hogy az A és B változók különböző értéket vesznek fel, feltéve hogy a B és C változók értéke megegyezik?

$$P(A \neq B \mid B = C) = \frac{P(A \neq B, B = C)}{P(B = C)} = \frac{\sum_a P(A = a, B = \neg a, C = \neg a)}{\sum_a \sum_b P(A = a, B = b, C = b)} =$$

$$= \frac{\sum_a P(A = a) P(B = \neg a \mid A = a) P(C = \neg a \mid A = a, B = \neg a)}{\sum_a P(A = a) \sum_b P(B = b \mid A = a) P(C = b \mid A = a, B = b)} =$$

$$= \frac{0,4 * 0,6 * 0,3 + 0,6 * 0,8 * 0,3}{0,4 * (0,4 * 0,5 + 0,6 * 0,3) + 0,6 * (0,8 * 0,3 + 0,2 * 0,1)} = \frac{0,216}{0,308} = 0,701$$

3. feladat

Ugyanezen Bayes-hálóval adott modellben mekkora a $P(C = i \mid A = B)$ valószínűség?

$$P(C = i \mid A = B) = \frac{P(C = i, A = B)}{P(A = B)} = \frac{\sum_a P(A = a, B = a, C = i)}{\sum_a \sum_c P(A = a, B = a, C = c)} =$$

$$= \frac{\sum_a P(A = a)P(B = a|A = a)P(C = i|A = a, B = a)}{\sum_a P(A = a)P(B = a|A = a) \sum_c P(C = c|A = a, B = a)} =$$

$$= \frac{\sum_a P(A = a)P(B = a|A = a)P(C = i|A = a, B = a)}{\sum_a P(A = a)P(B = a|A = a)} =$$

$$= \frac{0{,}4 * 0{,}4 * 0{,}5 + 0{,}6 * 0{,}2 * 0{,}9}{0{,}4 * 0{,}4 + 0{,}6 * 0{,}2} = \frac{0{,}188}{0{,}28} = 0{,}671$$