

Mesterséges intelligencia alapjai jegyzet

Molnár Antal Albert

2020. június 24.

Előszó

Ez a jegyzet a Debreceni Egyetemen a Kádek Tamás által oktatott, *INBPM0418E* tárgykódú *A mesterséges intelligencia alapjai* tárgyhoz nyújt némi segítséget a vizsgára készülő hallgatóknak.

A leírtakért semmilyen felelősséget nem tudok vállalni, hiszen még jómagam is csak ismerkedem a mesterséges intelligencia világával.

A jegyzet nagyrészt az órákon tárgyalt könyvből[1], valamint Várterész Magda előadásaiból[2] merít.

Tartalomjegyzék

1. Témakörök, melyeket mélységben ismerni kell	3
1.1. Ágens szemlélet	4
1.1.1. Az ágens fogalma	4
1.1.2. Az ágens jellemzése (teljesítmény, környezet, érzékelők, beavatkozók	5
1.2. Állapottér reprezentáció	7
1.2.1. Az állapottér fogalma	7
1.2.2. Az állapottérgráf	8
1.2.3. Költség és heurisztika fogalmak	9
1.3. Megoldáskereső algoritmusok	10
1.3.1. Fakereső algoritmusok	10
1.3.2. Gráfkereső algoritmusok	11
1.3.3. Szélességi kereső	12
1.3.4. Mélységi kereső	12
1.3.5. Visszalépéses kereső	13
1.3.6. Egyenletes költségű (optimális kereső)	13
1.3.7. Legjobbát először kereső	14
1.3.8. Az A* algoritmus	15
1.4. Kétszemélyes játékok	16
1.4.1. A játékok reprezentációja	16
1.4.2. A játékfa	17
1.4.3. Nyerő stratégia	17
1.5. Lépésajánló algoritmusok	18
1.5.1. MinMax módszer	18
1.5.2. NegaMax módszer	18
1.5.3. Alfa-béta nyesés	19
1.6. Élkonzisztencia algoritmusok	20
1.6.1. AC1	20
1.6.2. AC3	21
1.6.3. AC4	22
1.6.4. Visszalépéses kereső	23
2. Témakörök, melyekre rálátással kell rendelkezni	24
2.1. Következtetések ítéletlogikában	25
2.1.1. Rezolúciókalkulus ítéletlogikában	25
2.2. Döntési fák	27
2.2.1. Az ID3 algoritmus	27
2.3. Valószínűségi következtetés	28
2.3.1. Bayes hálók, Bayes tétel	30
2.3.2. Feltételes valószínűség számítása	32
2.4. Neurális hálók, kitekintés	33
2.4.1. Neurális hálók és mélytanulás	34

1. fejezet

Témakörök, melyeket mélységben ismerni kell

1.1. Ágens szemlélet

1.1.1. Az ágens fogalma

1. Definíció. Ágens. Egy **ágens** (agent) bármi lehet, amit úgy tekinthetünk, mint ami az **érzékelői** (sensors) segítségével érzékeli a **környezetét** (environment), és **beavatkozói** (actuators) segítségével megváltoztatja azt.

Az emberi ágensnek van szeme, füle és egyéb szervei az érzékelésre, és keze, lába, szája és egyéb testrészei a beavatkozásra.

A robotágens kamerákat és infravörös távolsági keresőket használ érzékelőként, és különféle motorokat beavatkozóként.

A szoftverágens billentyűleütéseket, fájl tartalmakat és hálózati adatcsomagokat fogad érzékelőinek bemeneteként, és képernyőn történő kijelzéssel, fájlok írásával, hálózati csomagok küldésével avatkozik be a környezetébe.

Azzal az általános feltételezéssel fogunk élni, hogy minden ágens képes saját akcióinak érzékelésére (de nem mindig látja azok hatását).

2. Definíció. Racionális ágens. Egy racionális ágens a legjobb várható kimenetel érdekében cselekszik.

Az ideális racionális ágens minden egyes észlelési sorozathoz a benne található tények és a beépített tudása alapján minden elvárható dolgot megtesz a teljesítménymérték maximalizálásáért.

3. Definíció. Érzékelés. Az érzékelés (percept) fogalmat használjuk az ágens érzékelő bemeneteinek leírására egy tetszőleges pillanatban.

4. Definíció. Ágens érzékelési sorozata. Egy ágens érzékelési sorozata (percept sequence) az ágens érzékeléseinek teljes története, minden, amit az ágens valaha is érzékelt. Általánosságban, egy adott pillanatban egy ágens cselekvése az addig megfigyelt teljes érzékelési sorozatától függhet. Ha az összes lehetséges érzékelési sorozathoz meg tudjuk határozni az ágens lehetséges cselekvéseit, akkor lényegében mindent elmondtunk az ágensről.

5. Definíció. Ágensfüggvény. Matematikailag megfogalmazva azt mondhatjuk, hogy az ágens viselkedését az ágensfüggvény (agent function) írja le, ami az adott érzékelési sorozatot egy cselekvésre képezi le.

6. Definíció. Ágensprogram. Egy mesterséges ágens belsejében az ágensfüggvényt egy ágensprogram (agent program) valósítja meg.

1. Megjegyzés. Különbség ágensfüggvény és ágensprogram között. Az ágensfüggvény egy absztrakt matematikai leírás, az ágensprogram egy konkrét implementáció, amely az ágens architektúráján működik.

1.1.2. Az ágens jellemzése (teljesítmény, környezet, érzékelők, beavatkozók)

7. Definíció. TKBÉ (PEAS) leírás. Egy ágens tervezése során az első lépésnek mindig a feladatkörnyezet lehető legteljesebb meghatározásának kell lennie.

Ágenstípus	Tel-jesítmény-mérték (Performance)	Környezet (Environment)	Beavatkozók (Actuators)	Érzékelők (Sensors)
Taxisofőr	Biztonságos, gyors, törvényes, kényelmes utazás, maximum haszon	Utak, egyéb forgalom, gyalogosok, ügyfelek	Kormány, gáz, fék, index, kürt, kijelző	Kamerák, hangradas, sebességmérő, GPS, kilométeróra, motorérzékelők, billentyűzet

8. Definíció. Teljesen megfigyelhető környezet.

Ha az ágens szenzorai minden pillanatban hozzáférést nyújtanak a környezet teljes állapotához, akkor azt mondjuk, hogy a környezet teljesen megfigyelhető.

9. Definíció. Determinisztikus (deterministic) vagy sztochasztikus (stochastic).

Amennyiben a környezet következő állapotát jelenlegi állapota és az ágens által végrehajtott cselekvés teljesen meghatározza, akkor azt mondjuk, hogy a környezet determinisztikus, egyébként sztochasztikus.

10. Definíció. Epizódszerű (episodic) vagy sorozatszerű (sequential) környezet.

Epizódszerű környezetben az ágens tapasztalata elemi "epizódokra" bontható. Minden egyes epizód az ágens észleléseiből és egy cselekvéséből áll. Nagyon fontos, hogy a következő epizód nem függ az előzőben végrehajtott cselekvésektől. Epizódszerű környezetekben az egyes epizódokban az akció kiválasztása csak az aktuális epizódtól függ.

11. Definíció. Statikus (static) vagy dinamikus (dynamic) környezet.

Ha a környezet megváltozhat, amíg az ágens gondolkodik, akkor azt mondjuk, hogy a környezet az ágens számára dinamikus; egyébként statikus.

12. Definíció. Diszkrét (discrete) vagy folytonos (continuous) környezet.

A diszkrét/folytonos felosztás alkalmazható a környezet állapotára, az időkezelés módjára, az ágens észleléseire, valamint cselekvéseire. Például egy diszkrét állapotú környezet, mint amilyen a sakkjáték, véges számú különálló állapottal rendelkezik. A sakkban szintén diszkrét az akciók és cselekvések halmaza. A taxivezetés folytonos állapotú és idejű probléma: a sebesség, a taxi és más járművek helye folytonos értékek egy tartományát járja végig a folytonos időben.

13. Definíció. Egyágenses (single agent) vagy többágenses (multiagent) környezet.

Az egyágenses és többágenses környezetek közötti különbségtétel egyszerűnek tűnhet. Például a kereszt-rejtvényt megfejtő ágens önmagában nyilvánvalóan egyágenses környezetben van, míg egy sakkozó ágens egy kétágensesben. Vannak azonban kényes kérdések. Először is: leírtuk azt, hogy egy entitás hogyan tekinthető ágensnek, ugyanakkor nem magyaráztuk meg, mely entitások tekintendők ágensnek. Egy A ágensnek (például a taxisornernek) egy B objektumot (egy másik járművet) ágensnek kell tekintenie, vagy egyszerűen egy sztochasztikusan viselkedő dolognak, a tengerparti hullámokhoz vagy a szélben szálló falevelekhez hasonlatosan? A választás kulcsa az, hogy vajon B viselkedése legjobban egy A viselkedésétől függő teljesítménymérték maximalizálásával írható le. Például a sakkban a B ellenfél saját teljesítménymértékét próbálja maximalizálni, amely – a sakk szabályainak következtében – A teljesítménymértékét minimalizálja. Így a sakk egy versengő (**competitive**) többágenses környezet. Másrészt, a taxi vezetési környezetben az ütközések elkerülése az összes ágens teljesítménymértékét maximálja, így az részben kooperatív (**cooperative**) többágenses környezet. Emellett részben versengő is, hiszen például csak egy autó tud egy parkolóhelyet elfoglalni. A többágenses környezetekben felmerülő ágenstervezési problémák gyakran egészen mások, mint egyágenses környezetekben. Többágenses környezetekben például a kommunikáció (communication) gyakran racionális viselkedésként bukkan fel; egyes részlegesen megfigyelhető versengő környezetekben a sztochasztikus viselkedés racionális, hiszen így elkerülhetők a megjósolhatóság csapdái.

1.2. Állapottér reprezentáció

1.2.1. Az állapottér fogalma

Az állapottér-reprezentáció az egyik leggyakrabban használt reprezentációs mód ami egy probléma formális megadására szolgál.

14. Definíció. Probléma. Egy **probléma** (problem) formális megragadásához az alábbi négy komponensre van szükség:

- **kiinduló állapot:** amiből az ágens kezdi a cselekvéseit,
- **cselekvések halmaza:** ágens rendelkezésére álló lehetséges cselekvések,
- **állapotátmenet-függvény:** visszaadja a rendezett \langle cselekvés, utódállapot \rangle párok halmazát (Egy alternatív megfogalmazás az **operátorok** egy halmaza, amelyeket egy állapotra alkalmazva lehet az utódállapotokat generálni.) lásd még: 15. Definíció
- **állapottér:** A kezdeti állapot és az állapotátmenet-függvény együttesen implicit módon definiálják a probléma **állapotterét:** azon állapotok halmazát, amelyek a kiinduló állapotból elérhetők.

15. Definíció. Operátor alkalmazási előfeltétel teszt. Ahhoz, hogy meggyőződjünk arról, hogy egy adott állapotra egy adott operátor alkalmazható, előbb meg kell vizsgálnunk, hogy az állapotra alkalmazható-e az operátor. Ezt a vizsgálódást operátor alkalmazási előfeltétel tesztnek nevezzük.

16. Definíció. Célteszt. A célteszt ellenőrzi a célfeltételek teljesülését egy adott állapotban.

17. Definíció. Állapottér-reprezentáció. Állapottér-reprezentáció alatt egy formális négyest értünk:

$$\langle \mathcal{A}, k, \mathcal{C}, \mathcal{O} \rangle,$$

ahol

- \mathcal{A} : az állapotok halmaza, $\mathcal{A} \neq \emptyset$
- $k \in \mathcal{A}$: a kezdőállapot
- $\mathcal{C} \subset \mathcal{A}$: a célállapotok halmaza
- \mathcal{O} : az operátorok halmaza, ahol $\mathcal{O} \neq \emptyset$
 - $\forall o \in \mathcal{O}$ egy operátor, azaz egy $o : \text{Dom}(o) \rightarrow \mathcal{A}$ függvény, ahol $\text{Dom}(o) = \{a \mid a \notin \mathcal{C} \wedge \text{előfeltétel}(o, a)\}$

18. Definíció. Közvetlen elérhetőség. Az $a \in \mathcal{A}$ állapotból az $a' \in \mathcal{A}$ állapot **közvetlenül elérhető**, ha van olyan $o_l \in \mathcal{O}$, amely esetén

$$a \in \text{dom}(o_l) \text{ és } o_l(a) = a'$$

és ezt $a \Rightarrow a'$ alakban jelöljük.

19. Definíció. Elérhetőség. Az $a \in \mathcal{A}$ állapotból az $a' \in \mathcal{A}$ állapot **elérhető** ($a \Rightarrow^* a'$), ha

- $a = a'$, vagy
- van olyan a_1, a_2, \dots, a_k állapotsorozat, hogy $a_1 = a, a_k = a'$ továbbá

$$a_i \Rightarrow a_{i+1} \text{ minden } i \in \{1, \dots, k-1\} \text{ esetén.}$$

1.2.2. Az állapottérgráf

20. Definíció. Állapottérgráf. Az állapottér egy gráfot alkot, amelynek csomópontjai az állapotok és a csomópontok közötti élek a cselekvések.

21. Definíció. Állapottér útja. Az **állapottér egy útja** az állapotok egy sorozata, amely állapotokat a cselekvések egy sorozata köt össze.

22. Definíció. Állapottér-reprezentációs gráf bonyolultsága. Egy állapottér-reprezentált probléma megoldásának sikerét jelentősen befolyásolja a reprezentációs gráf bonyolultsága:

- a csúcsok száma,
- az egy csúcsból kiinduló élek száma,
- a hurkok és körök száma és hossza.

Ezért célszerű minden lehetséges egyszerűsítést végrehajtani. Lehetséges egyszerűsítések:

- a csúcsok számának csökkentése — ügyes reprezentációval az állapottér kisebb méretű lehet;
- az egy csúcsból kiinduló élek számának csökkentése — az operátorok értelmezési tartományának alkalmas megválasztásával érhető el;
- a reprezentációs gráf fává alakítása — a hurkokat, illetve köröket „kiegyenesítjük”

Az állapottérgráf jellemzése

b : Az **elágazási tényező** (branching factor) a tetszőleges állapotból közvetlenül elérhető állapotok maximális száma.

$$\max\{\text{card}\{b : a \Rightarrow b\} : a \in \mathcal{A}\}.$$

d : A **legsekélyebb megoldás** a legkevesebb operátoralkalmazás segítségével elérhető célállapot eléréséhez szükséges operátoralkalmazások száma. (A legrövidebb megoldás hossza. A célállapotok minimális mélysége.) A legkisebb olyan i amely esetén van olyan állapotsorozat, hogy

$$a_0 \Rightarrow a_1, \quad a_1 \Rightarrow a_2 \quad \dots \quad a_{i-1} \Rightarrow a_i, \quad a_i \in \mathcal{C}.$$

m : A **csomópontok maximális mélysége**. A legnagyobb olyan i , amely esetén van olyan állapotsorozat, hogy

$$a_0 \Rightarrow a_1, \quad a_1 \Rightarrow a_2 \quad \dots \quad a_{i-1} \Rightarrow a_i.$$

1.2.3. Költség és heurisztika fogalmak

23. Definíció. Lépésköltség. Az x állapotból az y állapotba vezető cs cselekvés **lépésköltsége** (step cost) legyen $lk(x, cs, y)$. Tételezzük fel, hogy a lépésköltség nemnegatív.

24. Definíció. Útköltség-függvény. Egy **útköltség-függvény**, egy olyan függvény amely az állapottér minden útjához hozzárendel egy költséget.

25. Definíció. Megoldás. Egy út, amely a kiinduló állapotból egy célállapotba vezet.

26. Definíció. Optimális megoldás. A legkisebb útköltségű megoldás.

27. Definíció. Heurisztikus függvény. Egy $\langle \mathcal{A}, k, \mathcal{C}, \mathcal{O} \rangle$ állapottér-reprezentációhoz megadott heurisztika egy olyan

$$h : \mathcal{A} \mapsto \mathbb{N} \text{ függvény, melyre } \forall c \in \mathcal{C}h(c) = 0.$$

A heurisztika nem más mint egy becslés amely megmondja egy csúcsra nézve, hogy melyik gyermeke felé induljon tovább a keresésben. A heurisztikus kereső nem teljesen megbízható mivel a részfák még nincsenek legenerálva, azaz nem lehetünk biztosak benne.

Egyes heurisztikákat perfekt heurisztikának nevezünk és egy ilyen heurisztika legfeljebb $1 + n * d$ csúcsot generál le, azaz ilyen heurisztika eseté a csúcsok száma már csak lineáris függvénye a megoldás hosszának. (Perfekt heurisztika egyébként nem létezik, mert ha lenne akkor már előre ismernénk a megoldást - nem lenne értelme keresni.)

1.3. Megoldáskereső algoritmusok

1.3.1. Fakereső algoritmusok

Az általános fakeresési algoritmus informális leírása:

Algorithm 1: Általános fakeresési algoritmus informális leírása

```
1 function Fa-Kereses(probléma, stratégia):  
2   a probléma kezdeti állapotából kiindulva inicializáld a keresési fát  
3   loop do  
4     if nincs kifejtendő csomópont then return kudarc  
5     a stratégiának megfelelően válassz ki kifejtésre egy levélcsomópontot  
6     if a csomópont célállapotot tartalmaz then  
7       return a hozzá tartozó megoldás  
8     else  
9       fejtsd ki a csomópontot és az eredményül kapott csomópontokat, és add a keresési fához  
10    end if  
11  end loop  
12 end function
```

28. Definíció. Perem.

A legenerált, kifejtésre váró csomópontokat külön nyilvántartjuk, ez a **perem**. Ezt a gyakorlatban általában egy várakozási sorként (queue) szokás implementálni.

29. Definíció. Az algoritmusok hatékonyságának értékelése.

A problémamegoldó algoritmus kimenete vagy kudarc, vagy egy megoldás (egyes algoritmusok végtelen hurokba kerülhetnek és soha nem térnek vissza válasszal). Mi az algoritmusok hatékonyságát négyféle módon fogjuk értékelni:

- **Teljesség (completeness):** az algoritmus generáltan megtalál egy megoldást, amennyiben létezik megoldás?
- **Optimalitást (optimality):** a stratégia megtalálja az optimális megoldást?
- **Időigény (time complexity):** mennyi ideig tart egy megoldás megtalálása?
- **Tárigény (space complexity):** a keresés elvégzéséhez mennyi memóriára van szükség?

1.3.2. Gráfkereső algoritmusok

Eddig figyelmen kívül hagytuk a keresés egyik legfontosabb megoldandó problémáját: a már korábban egy másik úton megtalált és kifejtett állapotok ismételt kifejtéséből adódó időpazarlást. Néhány probléma esetén ezen lehetőség soha nem merül fel, mert az állapottér egy fa, és minden állapotba csak egyetlen módon lehet eljutni. A 8-királynő probléma hatékony megfogalmazása – amikor minden új királynőt a bal szélső szabad oszlopba helyezzük – nagyrészt épp ennek köszönheti hatékonyságát, vagyis hogy minden egyes állapotba csak egyetlen úton lehet eljutni. Ha a 8-királynő problémát úgy fogalmazzuk meg, hogy egy királynőt bármelyik oszlopban el lehet helyezni, akkor az n királynőt tartalmazó minden állapotot $n!$ különböző úton el lehet érni.

Számos problémánál azonban elkerülhetetlenek a megismételt állapotok. Ezek közé tartozik az összes olyan probléma, amelyben az operátorok reverzibilisek. Többek között ebbe a csoportba tartoznak az útkeresési problémák és a csúszó lapka fejtőre játékok. Az ezen problémákhoz tartozó keresési fák végtelenek, de ha a megismételt állapotok egy részét levágjuk, akkor a keresési fát véges méretűre vághatjuk, ezáltal a keresési fának csak az állapottér gráfot kifesztő részét generálva.

Az ismétlődő állapotok tehát a megoldható problémákat megoldhatatlan problémákká alakítják, amennyiben az algoritmus nem képes ezeket az állapotokat detektálni. A detektálás általában azt jelenti, hogy az új kifejtendő csomópontot a már kifejtett csomópontokkal hasonlítjuk össze. Egyezés esetén az adott csomóponthoz az algoritmus két utat talált, és valamelyiket eldobhatja.

Algorithm 2: Az általános gráfkereső algoritmus

```
1 function Graf-Kereses(probléma, perem) returns egy megoldás vagy kudarc:
2   zárt lista  $\leftarrow$  egy üres halmaz
3   perem  $\leftarrow$  Beszúr(Csomópontot-Létrehoz(Kiinduló-állapot[probléma]), perem)
4   loop do
5     if Üres?(perem) then return kudarc
6     csomópont  $\leftarrow$  Vedd-Az-Első-Element(perem)
7     if Cél-Teszt[probléma](Állapot[csomópont]) then return Megoldás(csomópont)
8     if Állapot[csomópont] nem eleme a zárt listának then
9       adjuk hozzá az Állapot[csomópont]-ot a zárt listához
10      perem  $\leftarrow$  Beszúr-Mind(Kifejt(csomópont, probléma), perem)
11    end if
12  end loop
13 end function
```

2. Megjegyzés. A zártak listáját érdemes egy hash táblával implementálni, hogy hatékonyan tudjuk ellenőrizni a már meglátogatott csomópontokat.

Jellemzés

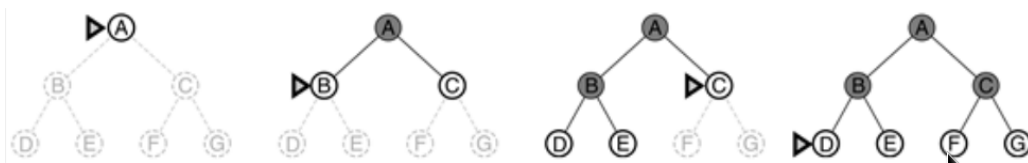
- **Időbonyolultság:** $1 + b + b^2 + \dots + b^{d+1} - b \in O(b^{d+1})$
- **Tárkonyolultság:** $1 + b + b^2 + \dots + b^{d+1} - b \in O(b^{d+1})$
- **Teljeség:** teljes, ha b véges,
- **Optimalitás:** optimális, ha minden költség egységnyi.

1.3.3. Szélességi kereső

A **szélességi keresés (breadth-first search)** egy egyszerű keresési stratégia, ahol először a gyökércsomópontot fejtjük ki, majd a következő lépésben az összes gyökércsomópontból generált csomópontot, majd azok követőit, stb.

Általánosságban a keresési stratégia minden adott mélységű csomópontot hamarabb fejt ki, mielőtt bármelyik, egy szinttel lejjebbi csomópontot kifejtené.

A szélességi keresést meg lehet valósítani a FA-KERESÉS algoritmussal egy olyan üres peremmel, amely egy először-be-először-ki (first-in-first-out – FIFO) sor, biztosítva ezzel, hogy a korábban meglátogatott csomópontokat az algoritmus korábban fejt ki.



1.1. ábra. Szélességi keresés egy egyszerű bináris fában

Jellemzés

- **Időbonyolultság:** $1 + b + b^2 + \dots + b^{d+1} - b \in O(b^{d+1})$
- **Tárbonyolultság:** $1 + b + b^2 + \dots + b^{d+1} - b \in O(b^{d+1})$
- **Teljesség:** teljes, ha b véges,
- **Optimalitás:** optimális, ha minden költség egységnyi.

1.3.4. Mélységi kereső

A **mélységi keresés (depth-first search)** mindig a keresési fa aktuális peremében a legmélyebben fekvő csomópontot fejt ki. A keresés azonnal a fa legmélyebb szintjére jut el, ahol a csomópontoknak már nincsenek követőik. Kifejtésüket követően kikerülnek a peremből és a keresés "visszalép" ahhoz a következő legmélyebben fekvő csomópontozhoz, amelynek vannak még ki nem fejtett követői.

Ez a stratégia egy olyan FA-KERESÉS függvénnyel implementálható, amelynek a sorbaállító függvénye az *utolsónak-be-elsőnek-ki (last-in-first-out, LIFO)*, más néven verem. A mélységi keresést szokás a FA-KERESÉS függvény alternatívájaként egy rekurzív függvénnyel is implementálni, amely a gyermekcsomópontokkal meghívja önmagát.

A mélységi keresés nagyon szerény tárigényű. Csak egyetlen, a gyökércsomóponttól egy levélcsomópontig vezető utat kell tárolnia, kiegészítve az út minden egyes csomópontja melletti kifejtetlen csomópontokkal. Egy kifejtett csomópont el is hagyható a memóriából, feltéve, hogy az összes leszármazottja meg lett vizsgálva. Egy b elágazási tényezőjű és m maximális mélységű állapottér esetén a mélységi keresés tárigénye $b \cdot m + 1$.

Jellemzés (fakeresőként)

- **Időbonyolultság:** $1 + b + b^2 + \dots + b^m \in O(b^m)$
- **Tárbonyolultság:** $1 + b \cdot m \in O(b \cdot m)$, feltéve, hogy minden olyan csomópontot elhagyunk, amely összes leszármazottja meg lett vizsgálva
- **Teljesség:** csak véges körmentes gráfban teljes
- **Optimalitás:** nem garantál optimális megoldásokat

1.3.5. Visszalépéses kereső

A mélységi keresés **visszalépéses keresésnek (backtracking search)** nevezett változata még kevesebb memóriát használ. A visszalépéses keresés az összes követő helyett egyidejűleg csak egy követőt generál. Minden részben kifejtett csomópont emlékszik, melyik követője jön a legközelebb. Így módon csak $O(m)$ memóriára van szükség, $O(b \cdot m)$ helyett. A visszalépéses keresés még egy memória- (és idő-) spóroló trükkhöz folyamodik. Az ötlet a követő csomópont generálása az aktuális állapot módosításával, anélkül hogy az állapotot átmásolnánk. Ezzel a memóriaszükséglet egy állapotra és $O(m)$ cselekvésre redukálódik. Ahhoz, hogy az ötlet működjön, amikor visszalépünk, hogy a következő követőt generáljuk, mindegyik módosítást vissza kell tudnunk csinálni. Nagy állapottérrel rendelkező problémák esetén, mint például robot-összeszerelés esetén, az ilyen módszerek lényegesek a sikerességhez.

A mélységi keresés hátrányos tulajdonsága, hogy egy rossz választással egy hosszú (akár végtelen) út mentén lefelé elakadhat, miközben például egy más döntés elvezetne a gyökérhez közeli megoldáshoz. A legrosszabb esetben a mélységi keresés a keresési fában az összes $O(b^m)$ csomópontot generálni fogja, ahol m a csomópontok maximális mélysége. Jegyezzük meg, hogy m sokkal nagyobb lehet, mint d (a legsekélyebb megoldás mélysége), és korlátlan fák esetén értéke végtelen.

3. Megjegyzés. A visszalépéses kereső használatával a mélységi keresés tárbonyolultsága csökkenthető tovább.

Jellemzés

- **Időbonyolultság:** $1 + b + b^2 + \dots + b^m \in O(b^m)$
- **Tárbonyolultság:** $1 + m \in O(m)$
- **Teljesség:** csak véges körmentes gráfban teljes
- **Optimalitás:** nem garantál optimális megoldásokat

1.3.6. Egyenletes költségű (optimális kereső)

A szélességi keresés optimális, ha minden lépés költsége azonos, mert mindig a legsekélyebb ki nem fejtett csomópontot fejt ki. Egyszerű általánosítással egy olyan algoritmust találhatunk ki, amely tetszőleges lépésköltség mellett optimális. Az **egyenletes költségű keresés (uniform cost search)** mindig a legkisebb útköltségű n csomópontot fejt ki először, nem pedig a legkisebb mélységű csomópontot. Egyszerűen belátható, hogy a szélességi keresés is egyenletes költségű keresés, amennyiben minden lépésköltség azonos.

Az egyenletes költségű keresés nem foglalkozik azzal, hogy *hány* lépésből áll egy bizonyos út, hanem csak az összköltségükkel törődik. Emiatt mindig végtelen hurokba kerül, ha egy csomópont kifejtése zérus költségű cselekvéshez és ugyanahhoz az állapothoz való visszatérést eredményez (például a NOOP cselekvés).

Tegyük fel, hogy előállítjuk az m csomópontot az n csomópont gyermekekeként és be szeretnénk szűrni az adatbázisba, ahol m már szerepel. Ennek az útköltségét jelöljük $g(m)$ -mel, az újonnan beszűrni kívánt csomópont útköltségét pedig $g(n) + \text{költség}(\text{operátor})(n)$ -nel. Ekkor ha

- $g(m) \leq g(n) + \text{költség}(\text{operátor})(n)$, akkor az m csomópontot már sikerült ezelőtt kisebb költséggel előállítani, és nem cseréljük le.
- $g(m) > g(n) + \text{költség}(\text{operátor})(n)$, akkor az új csúcs optimálisabb (tehát nem nagyobb költségű), lecseréljük.

A teljességet csak úgy garantálhatjuk, hogy minden lépés költsége egy kis pozitív ϵ konstansnál nagyobb, vagy azzal egyenlő. Ez a feltétel egyben az optimalitás elégséges feltétele is.

Ez azt jelenti, hogy egy út költsége az út mentén mindig növekszik. Ebből a tulajdonságból látszik, hogy az algoritmus a csomópontokat mindig a növekvő útköltség függvényében fejt ki. Azaz az első kifejtésre kiválasztott célcsomópont egyben az optimális megoldás is (emlékezzünk arra, hogy a FA-KERESÉS a célállapottesztet csak a kifejtésre megválasztott csomópontokra alkalmazza).

Ha C^* az optimális út útköltsége, és minden útköltség legalább ϵ , akkor az időbonyolultság $O(b^{1+(C^*/\epsilon)})$.

Mivel az egyenletes költségű kereső mindent csomópontot tárol mindaddig, amíg megoldást nem talál, a tárigénye a szélességi keresőnél is rosszabb.

Akkor használatos, mikor az útköltségek nem azonosak, amikor azonosak, akkor BFS-sel is jobban járunk.

Jellemzés

- **Időbonyolultság:** lehet rosszabb, $O(b^d)$, mert felkutathat nagy fákat; $O(b^{1+(C^*/e)})$
- **Tárbonyolultság:** $O(b^{1+(C^*/e)})$
- **Teljesség:** ha minden lépésköltség egy kis pozitív szám
- **Optimalitás:** ha minden lépésköltség egy kis pozitív szám

1.3.7. Legjobbat először kereső

Informált keresési módszer.

A **mohó legjobbat-először keresés (greedy best-first search)** azt a csomópontot fejt ki a következő lépésben, amelyiknek az állapotát a legközelebbinek ítéli a célállapothoz, abból kiindulva, hogy így gyorsan megtalálja a megoldást. A csomópontokat az algoritmus tehát az $f(n) = h(n)$ heurisztikus függvénnyel értékeli ki.

A legjobbat-először keresés az általános FA-KERESÉS vagy GRÁF-KERESÉS algoritmusok olyan speciális esete, ahol egy csomópont kifejtésre való kiválasztása egy $f(n)$ kiértékelő függvénytől (evaluation function) függ. A heuristikáját nézi a csomópontnak, és a kiterjesztett szülőcsomópont, amit lát az utódokon heurisztika az alapján választja ki a legkisebb heurisztikával rendelkező gyermeket, és fogja azt kiterjeszteni.

Hagyományosan a legkisebb értékű csomópontot választjuk kifejtésre, mert a kiértékelő függvény a céltól való távolságot méri. A legjobbat-először keresés az eddigi általános keresési eljárások keretein belül egy prioritási sor segítségével implementálható, ami egy olyan adatstruktúra, mely a peremet a növekvő f -értékek szerint rendezi.

Jellemzés

- **Időbonyolultság:** $O(b^m)$
- **Tárbonyolultság:** $O(b^m)$
- **Teljesség:** nem teljes
- **Optimalitás:** nem optimális

4. Megjegyzés. Az idő- és tárbonyolultság nagyban függ a heurisztikus függvény minőségétől.

1.3.8. Az A* algoritmus

Informált keresési módszer.

A legjobbat-először keresés leginkább ismert változata az A* keresés. A csomópontokat úgy értékeli ki, hogy összekombinálja $g(n)$ értékét – az aktuális csomópontig megtett út költsége – és $h(n)$ értékét – vagyis az adott csomóponttól a célhoz vezető út költségének becslőjét:

$$f(n) = g(n) + h(n).$$

Mivel $g(n)$ megadja a kiinduló csomóponttól az n csomópontig számított útköltséget, és $h(n)$ az n csomóponttól a célcsomópontba vezető legolcsóbb költségű út költségének becslője, így az alábbi összefüggést kapjuk:

$$f(n) = \text{a legolcsóbb, az } n \text{ csomóponton keresztül vezető megoldás becsült költsége.}$$

Így amennyiben a legolcsóbb megoldást keressük, ésszerű először a legkisebb $g(n) + h(n)$ értékkel rendelkező csomópontot kifejezni. Ezen stratégia kellemes tulajdonsága, hogy ez a stratégia több mint ésszerű: amennyiben a h függvény eleget tesz bizonyos feltételeknek, az A* keresés teljes és optimális.

Jellemzés

- **Időbonyolultság:** $O(b^m)$
- **Tárbonyolultság:** $O(b^m)$
- **Teljesség:** teljes
- **Optimalitás:**
 - fakesésőként ha $h(n)$ elfogadható heurisztika
 - gráfkeresőként optimális, ha $h(n)$ konzisztens (vagyis monoton), valamint elfogadható heurisztika, azaz soha nem becsül felül (ez következik a konzisztenciából)

30. Definíció. Heurisztikus függvény. Egy $\langle \mathcal{A}, k, \mathcal{C}, \mathcal{O} \rangle$ állapotér-reprezentációhoz megadott heurisztika egy olyan

$$h : \mathcal{A} \mapsto \mathbb{N} \text{ függvény, melyre } \forall c \in \mathcal{C} h(c) = 0.$$

A heurisztika nem más mint egy becslés amely megmondja egy csúcsra nézve, hogy melyik gyermeke felé induljon tovább a keresésben. A heurisztikus kereső nem teljesen megbízható mivel a részfák még nincsenek legenerálva, azaz nem lehetünk biztosak benne.

Egyes heurisztikákat perfekt heurisztikának nevezünk és egy ilyen heurisztika legfeljebb $1 + n * d$ csúcsot generál le, azaz ilyen heurisztika eseté a csúcsok száma már csak lineáris függvénye a megoldás hosszának. (Perfekt heurisztika egyébként nem létezik, mert ha lenne akkor már előre ismernénk a megoldást - nem lenne értelme keresni.)

1. Tétel. $h(n)$ elfogadható heurisztika, ha soha nem becsli felül az n csomópontot keresztül a célállapothoz vezető út útköltségét.

31. Definíció. Konzisztencia. Azt mondjuk, hogy a $h(n)$ heurisztikus függvény konzisztens, ha minden n csomópontra és annak egy tetszőleges a cselekvéssel generált minden n' utódcsomópontjára az n csomóponttól elért cél becsült költsége nem kisebb, mint az n' -be kerülés lépésköltsége és az n' csomóponttól elért cél becsült költsége:

$$h(n) \leq c(n, a, n') + h(n').$$

Ez az általános háromszög egyenlőtlenség (triangle inequality) egy formája, amely azt fejezi ki, hogy egy háromszög egy oldala sem lehet hosszabb, mint a két másik oldal összege.

1.4. Kétszemélyes játékok

1.4.1. A játékok reprezentációja

A továbbiakban a

- teljesen megfigyelhető,
- véges,
- determinisztikus,
- kétszemélyes,
- zérusösszegű

játékokkal foglalkozunk.

32. Definíció. Játék reprezentációja. Egy játék reprezentációja megadható a

$$\langle \mathcal{B}, b_0, \mathcal{J}, \mathcal{V}, \hat{v}, \mathcal{L} \rangle$$

rendezett hatossal, ahol:

- \mathcal{B} : a játékállások halmaza,
- b_0 : a kezdőállás, ahol $b_0 \in \mathcal{B}$,
- \mathcal{J} : a játékosok halmaza, ahol $\text{card } \mathcal{J} = 2$,
- \mathcal{V} : a végállások halmaza, ahol $\mathcal{V} \subseteq \mathcal{B}$,
- \hat{v} : egy $\mathcal{V} \rightarrow \{-1, 0, 1\}$ függvény,
- \mathcal{L} : a lépések halmaza.

33. Definíció. Nyertes. Végállásban a \hat{v} függvény határozza meg, hogy melyik játékos nyert:

$$\hat{v}(b) = \begin{cases} 1 & \text{ha } b \text{ állásban a következő játékos nyer} \\ 0 & \text{ha } b \text{ állásban a következő játékos veszít} \\ -1 & \text{egyébként (döntetlen esetén)} \end{cases}$$

34. Definíció. Lépés. Minden $l \in \mathcal{L}$ egy $\mathcal{B} \rightarrow \mathcal{B}$ parciális függvény.

35. Definíció. Játék állapottere. Legyen $\langle \mathcal{B}, b_0, \mathcal{J}, \mathcal{V}, \hat{v}, \mathcal{L} \rangle$ egy játék reprezentációja. Ekkor a **játék állapottere** $\langle \mathcal{A}, a_0, \mathcal{C}, \mathcal{O} \rangle$ definiálható a következőképpen:

- $\mathcal{A} = \mathcal{B} \times \mathcal{J}$;
- $a_0 = \langle b_0, j_0 \rangle$, ahol $j_0 \in \mathcal{J}$ a kezdőjátékos;
- $\mathcal{C} = \{ \langle b, j \rangle : \langle b, j \rangle \in \mathcal{A} \wedge b \in \mathcal{V} \}$
- $\mathcal{O} = \{ o_l : l \in \mathcal{L} \}$, ahol $o_l : \mathcal{A} \rightarrow \mathcal{A}$, úgy hogy
 - $\text{dom}(o_l) = \{ \langle b, j \rangle : b \in \text{dom}(l) \}$
 - $o_l(\langle b, j \rangle) = \langle l(b), j' \rangle$
 - $j' \in (\mathcal{J} \setminus \{j\})$

36. Definíció. Közvetlen elérhetőség. Az $a \in \mathcal{A}$ állapotból az $a' \in \mathcal{A}$ állapot **közvetlenül elérhető**, ha van olyan $o_l \in \mathcal{O}$, amely esetén

$$a \in \text{dom}(o_l) \text{ és } o_l(a) = a'$$

és ezt $a \Rightarrow a'$ alakban jelöljük.

37. Definíció. Elérhetőség. Aza $a \in \mathcal{A}$ állapot az $a' \in \mathcal{A}$ állapot **elérhető** ($a \Rightarrow^* a'$), ha

- $a = a'$, vagy
- van olyan a_1, a_2, \dots, a_k állapotsorozat, hogy $a_1 = a, a_k = a'$ továbbá

$$a_i \Rightarrow a_{i+1} \text{ minden } i \in \{1, \dots, k-1\} \text{ esetén.}$$

1.4.2. A játékfa

38. Definíció. Játékfa. Legyen $\langle \mathcal{B}, b_0, \mathcal{J}, \mathcal{V}, \hat{v}, \mathcal{L} \rangle$ egy játék reprezentációja és $j_0 \in \mathcal{J}$ a kezdőjátékos. Ekkor a játékfa olyan fa, melynek csúcsaihoz a játék állapotait rendeljük:

- a fa gyökere a $\langle a_0, j_0 \rangle$ állapottal címkézett csúcs,
- a fa levélelemei olyan $\langle b, j \rangle$ állapottal címkézett csúcsok, ahol $b \in \mathcal{V}$
- a fa $\langle b, j \rangle$ állapottal címkézett nem levélcsúcsának gyermekeit olyan $\langle b', j' \rangle$ címkéjű csúcsok alkotják, ahol $\langle b, j \rangle \Rightarrow \langle b', j' \rangle$

5. Megjegyzés. A játékfa a játék állapotterének gráfját fává egyenesíti ki, egy-egy állapot a fában több csúcs címkéjeként is szerepelhet.

1.4.3. Nyerő stratégia

39. Definíció. Játsszma. Egy **játsszma** egy olyan lépéssorozat, amely a $\langle b_0, j_0 \rangle$ kezdőállapotból valamely $\langle b, j \rangle \in \mathcal{C}$ célállapotba vezet. (Egy a gyökértől valamely levélcsúsig vezető út a játékfában).

40. Definíció. Stratégia. Játékterv, ami minden olyan állásban, amikor ő következik, megmondja a játékosnak, hogy mit lépjen.

A $j \in \mathcal{J}$ játékos **stratégiája** olyan $S_j : \{\langle b, j \rangle : \langle b, j \rangle \in \mathcal{A}\} \rightarrow \mathcal{O}$ leképezés (döntési terv), amely j számára előírja, hogy a játék azon állapotaiban, ahol j a lépni következő játékos, a megtehető lépések közül melyiket lépje meg.

41. Definíció. Nyerő stratégia.

A $j \in \mathcal{J}$ játékos egy S_j stratégiája **nyerő stratégia**, ha minden a stratégia mellett lejátszható játszmában j nyer.

Ha döntetlen állhat elő – $0 \notin \text{rng}(\hat{v})$ –, akkor az egyik játékos rendelkezik nyerő stratégiával. A nyerő stratégiával rendelkező játékos $w(a_0, j_0)$:

$$w(b, j) = \begin{cases} j & \text{ha } b \in \mathcal{V} \text{ és } \hat{v}(b) = 1 \\ j' & \text{ha } b \in \mathcal{V} \text{ és } \hat{v}(b) = -1 \\ j & \text{ha van olyan } \langle b', j' \rangle \text{ hogy } \langle b, j \rangle \rightarrow \langle b', j' \rangle \text{ és } sw(b', j') = j \\ j' & \text{egyébként,} \end{cases}$$

ahol $j' \in (\mathcal{J} \setminus \{j\})$.

6. Megjegyzés. Ha a játékban van döntetlen végállás, akkor az egyik játékosnak van garantáltan nem veszteső stratégiája.

1.5. Lépésajánló algoritmusok

42. Definíció. Hasznosságfüggvény.

A **hasznosságfüggvény** egy becslés, de elvárjuk, hogy végállásban pontos legyen. Formálisan:

$$h : \mathcal{A} \rightarrow [-m, m] \text{ ahol } m \in \mathbb{R}^+ \text{ továbbá ha } b \in \mathcal{V} \text{ akkor } h(b, j) = m \cdot \hat{v}(b).$$

Ha $h(b, j) > h(b', j)$ akkor a b állás a j (lépni következő) játékos számára kedvezőbb, mint a b' állás.

Legyen h egy hasznosságfüggvény, ekkor h_i az $i \in \mathcal{J}$ játékos hasznosságfüggvénye, ha

$$h_i(a, j) = \begin{cases} h(a, j) & \text{ha } i = j \\ -h(a, j) & \text{egyébként.} \end{cases}$$

1.5.1. MinMax módszer

Az $b \in \mathcal{B}$ állás hasznossága $t \in \mathcal{J}$ támogatott játékos számára ha $j \in \mathcal{J}$ játékos következik lépni, $k \in \mathbb{N}$ mélységi korláttal rekurzívan a következőképpen számítható:

$$f(b, j, t, k) = \begin{cases} h_t(b, j) & \text{ha } b \in \mathcal{V}, \\ h_t(b, j) & \text{ha } k = 0, \\ \min\{f(b', j', t, k-1) : \langle b, j \rangle \Rightarrow \langle b', j' \rangle\} & \text{ha } t \neq j, \\ \max\{f(b', j', t, k-1) : \langle b, j \rangle \Rightarrow \langle b', j' \rangle\} & \text{ha } t = j. \end{cases}$$

A MinMax algoritmus az optimális döntést az aktuális állapotból számítja ki, felhasználva az egyes követő állapotok minimax értékeinek kiszámítására a definiáló egyenletekből közvetlenül származtatott, egyszerű rekurzív formulát. A rekurzió egészen a falevelekig folytatódik, majd a minimax értékeket a fa mentén visszafelé terjesztjük, ahogy a rekurzió visszalép. Bizonyos esetben például az algoritmus először rekurzív módon leereszkedik a három bal alsó csomóponthoz, a HASZNOSSÁG függvénnyel kiszámítva. Majd az algoritmus előveszi ezen értékek minimumát és ezt adja vissza, ahogy a szülő csomópontához visszatér.

Minden olyan ponton, ahol a támogatott játékos van döntési helyzetben, akkor ő maga számára a lehető legjobbat akarja, ezért a közvetlenül elérhető játé

Az ellenfele önmagának akarja a legjobbat, viszont ami az ellenfélnek a legjobb, az a támogatott játékos számára a legrosszabb, tehát ezért van ebben az esetben minimum keresés ebben az implementációban.

1.5.2. NegaMax módszer

Az $b \in \mathcal{B}$ állás hasznossága $j \in \mathcal{J}$ (lépni következő) játékos számára $k \in \mathbb{N}$ mélységi korláttal rekurzívan a következőképpen számítható:

$$\hat{f}(b, j, k) = \begin{cases} h(b, j) & \text{ha } b \in \mathcal{V}, \\ h(b, j) & \text{ha } k = 0, \\ \max\{-\hat{f}(b', j', k-1) : \langle b, j \rangle \Rightarrow \langle b', j' \rangle\} & \text{egyébként.} \end{cases}$$

A NegaMax algoritmus esetében is hasonlóan kell gondolkodnunk, mint a MinMax módszernél. Kiegészítés: A minimax algoritmus alkalmazásának egyik nehézsége, hogy egy játékhoz kétfajta heurisztikát is ki kell találni. Márpedig könnyű észre venni, hogy a két heurisztika között erős összefüggés van. Az összefüggés egyszerű szavakkal így írható le: A játék egy állapota az egyik játékosnak minél jobb, a másiknak annál rosszabb. Azaz az egyik játékos heurisztikája nyugodtan lehet a másik játékos heurisztikájának ellentéte (negáltja), vagyis -1 -szerese. Tehát, elég lesz 1 db heurisztika.

1.5.3. Alfa-béta nyesés

A gyakorlatban a MinMax algoritmus javított változatát szokás használni, az alfa-béta nyesést.

43. Definíció. Az ajánlott lépés. A $b \in \mathcal{B}$ állásban **ajánlott lépés** a $j \in \mathcal{J}$ (lépni következő) játékos számára olyan $l \in \mathcal{L}$ lépés, amely esetén

- $b \in \text{dom } l$,
- $o_l(\langle b, j \rangle) = \langle b', j' \rangle$, és
- $f(b, j, j, k) = f(b', j', j, k - 1)$.

A nyesés vagy metszés lehetővé teszi számunkra, hogy figyelmen kívül hagyjuk a keresési fa azon részeit, amelyek nincsenek befolyással a végső választásra.

A heurisztikus kiértékelő függvények lehetővé teszik, hogy kimerítő keresés nélkül meg tudjuk becsülni egy adott állapot valódi hasznosságát. Tehát a játékfa nagyobb részét is kihagyhatjuk adott esetben.

A konkrét vizsgált technika az alfa-béta nyesés (alpha-beta pruning). Ha ezt egy standard minimax fára alkalmazzuk, ugyanazt az eredményt adja vissza, mint a minimax, a döntésre hatással nem lévő ágakat azonban lenyeli. A célunk az, hogy meg tudjuk határozni a minimax döntést anélkül, hogy megkellene vizsgálnunk az összes levélcsomópontot.

- α = az út mentén tetszőleges döntési pontban a MAX számára eddig megtalált legjobb (azaz a legmagasabb értékű) választás értéke.
- β = az út mentén tetszőleges döntési pontban a MIN számára eddig megtalált legjobb (azaz a legkisebb értékű) választás értéke.

1.6. Élkonzisztencia algoritmusok

44. Definíció. Legyen \mathcal{V} változók egy tetszőleges halmaza, \mathcal{D} pedig egy olyan leképezés, amely minden $x \in \mathcal{V}$ változóhoz egy $\mathcal{D}(x)$ -el jelölt halmazt (tartományt) rendel.

Tetszőleges két különböző $x \in \mathcal{V}$ és $y \in \mathcal{V}$ változó közti bináris kényszer egy $\mathcal{D}(x)$ és $\mathcal{D}(y)$ feletti bináris reláció:

$$R_{x,y} \subseteq \mathcal{D}(x) \times \mathcal{D}(y).$$

Jelöljük $R_{x,y}^{-1}$ formában az $R_{x,y}$ reláció inverzét:

$$R_{x,y}^{-1} = \{(b, a) : (a, b) \in R_{x,y}\}.$$

45. Definíció. Véges, bináris kényszerkielégítési probléma. Egy **véges, bináris kényszerkielégítési probléma** alatt olyan $\langle \mathcal{V}, \mathcal{D}, \mathcal{C} \rangle$ rendezett hármast értünk, ahol

- \mathcal{V} változók egy véges, nemüres halmaza
- $\mathcal{D}(x)$ minden $x \in \mathcal{V}$ változó esetén egy véges halmaz(tartomány)
- \mathcal{C} a \mathcal{V} -beli változók feletti bináris kényszerek halmaza, úgy hogy
 - ha $R_{x,y} \in \mathcal{C}$ és $R'_{x,y} \in \mathcal{C}$ akkor $R_{x,y} = R'_{x,y}$.
 - ha $R_{x,y} \in \mathcal{C}$ akkor $R_{y,x} \in \mathcal{C}$ és $R_{y,x} = R_{x,y}^{-1}$.

46. Definíció. Gráf reprezentáció. Egy $\langle \mathcal{V}, \mathcal{D}, \mathcal{C} \rangle$ (nem feltétlenül véges) **bináris kényszerkielégítési probléma gráf reprezentációja** egy olyan $\langle N, E \rangle$ pár, ahol:

- $N = \{n_v : v \in \mathcal{V}\}$ alkotja a gráf csúcsait,
- $E = \{(n_x, n_y) : R_{x,y} \in \mathcal{C}\}$ alkotja a gráf irányított éleit.

7. Megjegyzés. Mivel $R_{x,y}$ és $R_{y,x}$ bináris kényszerek ugyanazt fejezik ki, ezért az irányított élek helyett irányítatlan éleket tartalmazó gráfot szokás használni.

Annak, hogy $R_{x,y}$ és $R_{y,x}$ együtt szerepel a kényszerek között, a később bemutatott algoritmusokban lesz jelentősége.

47. Definíció. Nem bináris kényszerek.

Unáris kényszerek: Az unáris kényszerek egyetlen változó értékét korlátoznák. A helyett, hogy ezt kényszerként jelenítenénk meg, kifejezhetjük azzal, hogy a változóhoz rendelt tartomány elemeiként eleve csak az unáris kényszernek megfelelő értékeket választunk.

Magasabb rendű kényszerek: A magasabb rendű kényszerek mindig kiválthatók bináris kényszerekkel újabb változók bevezetése mellett.

48. Definíció. Kényszerek terjesztése. A **kényszerek terjesztése** során egy tekintett változó értékére vonatkozó megszorítás következményeit a vele kényszerek útján kapcsolatban álló változók értékeire vonatkozóan is kiterjesztjük, ezen változók értékeire újabb megszorításokat alkalmazva.

49. Definíció. Élkonzisztencia algoritmusok. Az **élkonzisztencia algoritmusok** feladata, a kényszerek terjesztésének hatékony megvalósítása. Ezen algoritmusok egyaránt alkalmazhatóak a keresés megkezdése előtt a probléma méretét csökkentő előfeldolgozó lépésként, vagy akár a keresés közben is. Ezek közül mi az előbbi lehetőséget vizsgáljuk meg.

1.6.1. AC1

Az AC-1 (gyakorlatban nem használt) algoritmus egyszerű naiv megközelítés segítségével mutatja be a kényszerek terjesztésének ötletét.

50. Definíció. Felülvizsgálat. Az $R_{x,y}$ **kényszer felülvizsgálata** során a $\langle \mathcal{V}, \mathcal{D}, \mathcal{C} \rangle$ bináris kényszereket tartalmazó véges kényszerkielégítési problémát a $\langle \mathcal{V}, \mathcal{D}', \mathcal{C}' \rangle$ problémával helyettesítjük, ahol

$$\mathcal{D}' = \begin{cases} \{a : a \in \mathcal{D}(x) \wedge b \in \mathcal{D}(y) \wedge \langle a, b \rangle \in R_{x,y}\} & \text{ha } z = x \\ \mathcal{D}(z) & \text{egyébként.} \end{cases}$$

Algorithm 3: AC-1

```

1 function AC1( $\langle \mathcal{V}, \mathcal{D}, \mathcal{C} \rangle$ ):
2   repeat
3      $\mathcal{D}' \leftarrow \mathcal{D}$ 
4     forall  $R_{x,y} \in \mathcal{C}$  do
5        $\mathcal{D} \leftarrow \text{revise}(R_{x,y}, \langle \mathcal{V}, \mathcal{D}, \mathcal{C} \rangle)$ 
6     end forall
7   until  $\mathcal{D} = \mathcal{D}'$ 
8   return  $\langle \mathcal{V}, \mathcal{D}, \mathcal{C} \rangle$ 
9 end function

```

1.6.2. AC3

Az AC-3 algoritmus (Mackworth) egy már a gyakorlatban is használható algoritmus, amely egy sorban tartja nyilván azokat a kényszereket, amelyeket felül kell vizsgálni.

Ha $R_{x,y}$ felülvizsgálat során az x változó tartománya megváltozik, akkor minden olyan kényszer ismét bekerül a sorba, amely $R_{y,x}$ alakú.

Az algoritmus akkor fejeződik be, mikor elfogytak a sorból a kényszerek.

Az algoritmus időbonyolultsága $\mathcal{O}(n^2 \cdot d^3)$.

Algorithm 4: AC-3

```

1 function AC3( $\langle \mathcal{V}, \mathcal{D}, \mathcal{C} \rangle$ ):
2   while  $W \neq \emptyset$  do
3     remove an  $R_{x,y}$  constraint from  $W$ 
4      $\mathcal{D}' \leftarrow \mathcal{D}$ 
5      $\mathcal{D} \leftarrow \text{revise}(R_{x,y}, \langle \mathcal{V}, \mathcal{D}, \mathcal{C} \rangle)$ 
6     if  $\mathcal{D}' \neq \mathcal{D}$  then
7       if  $\mathcal{D}(x) = \emptyset$  then
8         return failure
9       else
10        forall  $R_{u,w} \in \{R_{u,w} : R_{u,w} \in \mathcal{C} \wedge w = x\}$  do
11           $W \leftarrow W \cup \{R_{u,w}\}$ 
12        end forall
13      end if
14    end if
15  end while
16  return  $\langle \mathcal{V}, \mathcal{D}, \mathcal{C} \rangle$ 
17 end function

```

1.6.3. AC4

Az AC-4 algoritmus (Mohr és Henderson) két előre elkészített adatszerkezettel dolgozik:

- minden $R_{x,y}$ kényszer és $v_x \in \mathcal{D}(x)$ értékhez egy számlálót rendelünk:

$$C_{x,v_x,y} = \text{card}\{v_y : v_y \in \mathcal{D}(y) \wedge \langle v_x, v_y \rangle \in R_{x,y}\},$$

- az x változó minden $v_x \in \mathcal{D}(x)$ értékéhez egy halmazt rendelünk:

$$S_{x,v_x} = \{\langle y, v_y \rangle : v_y \in \mathcal{D}(y) \wedge \langle v_y, v_x \rangle \in R_{y,x}\},$$

amely megmutatja, hogy mely y változók mely $v_y \in \mathcal{D}(y)$ értékeihez számoltuk hozzá v_x -et.

Jelölje $\mathcal{D}_{-\langle x, v_x \rangle}$ azt a leképezést, amely \mathcal{D} leképezéstől annyiban különbözik, hogy $v_x \notin \mathcal{D}(x)$, vagyis x nem veheti fel v_x értéket:

$$\mathcal{D}_{-\langle x, v_x \rangle}(z) = \begin{cases} \mathcal{D}(x) \setminus \{v_x\} & \text{ha } z = x, \\ \mathcal{D}(z) & \text{egyébként.} \end{cases}$$

Algorithm 5: AC-4 inicializálás

```

1 function AC4-initialize( $\langle \mathcal{V}, \mathcal{D}, \mathcal{C} \rangle$ ):
2   Compute  $S$ 
3   Compute  $C$ 
4   forall  $\langle x, v_x \rangle \in \{\langle x, v_x \rangle : C_{x,v_x,y} = 0\}$  do
5      $\mathcal{D} \leftarrow \mathcal{D}_{-\langle x, v_x \rangle}$ 
6   end forall
7   return  $S, C, \mathcal{D}$ 
8 end function
```

Algorithm 6: AC-4

```

1 function AC4( $\langle \mathcal{V}, \mathcal{D}, \mathcal{C} \rangle$ ):
2    $S, C, \mathcal{D} \leftarrow \text{AC4-initialize}(\langle \mathcal{V}, \mathcal{D}, \mathcal{C} \rangle)$ 
3    $W \leftarrow \{\langle x, v_x \rangle : C_{x,v_x,y} = 0\}$ 
4   while  $W \neq \emptyset$  do
5     remove an  $\langle x, v_x \rangle$  pair from  $W$ 
6     forall  $\langle y, v_y \rangle \in S_{x,v_x}$  do
7        $C_{y,v_y,x} \leftarrow C_{y,v_y,x} - 1$ 
8       if  $C_{y,v_y,x} = 0$  and  $v_y \in \mathcal{D}_y$  then
9          $\mathcal{D} \leftarrow \mathcal{D}_{-\langle y, v_y \rangle}$ 
10        insert the  $\langle y, v_y \rangle$  pair into  $W$ 
11      end if
12    end forall
13  end while
14  return  $S, C, \mathcal{D}$ 
15 end function
```

Az **AC-4 algoritmus** egy váltótó-érték párokat tartalmazó munkahalmaz segítségével végzi el feladatát. Azon értékek, amelyek által C számlálója 0-ra csökken törölni kell a változó tartományából. A munkahalmazba a törtölt értékek kerülnek, mert a tőlük függő értékek számlálóját csökkenteni kell. Ezek gyors felderítésében segít S .

8. Megjegyzés. Az AC-4 algoritmus időbonyolultsága: $\mathcal{O}(n^2 \cdot d^2)$. Ennek ellenére valódi problémák esetében sok esetben az AC-3 algoritmus teljesít jobban (Richard J. Wallace).

1.6.4. Visszalépéses kereső

```
function VISSZALÉPÉSÉS-KERESÉS(csp) returns egy megoldás vagy kudarc  
  return REKURZÍV-VISSZALÉPÉSÉS({}, csp)  
  
function REKURZÍV-VISSZALÉPÉSÉS(hozzárendelések, csp) returns egy megoldás vagy kudarc  
  if hozzárendelések teljes then return hozzárendelések  
  var  $\leftarrow$  HOZZÁRENDELETLEN-VÁLTOZÓ-KIVÁLASZTÁSA(VÁLTOZÓK[csp], hozzárendelések, csp)  
  for each érték in TARTOMÁNY-ÉRTÉKEK-SORRENDEZÉSE(var, hozzárendelések, csp) do  
    if érték konzisztens a hozzárendelések-kel KÉNYSZEREK(csp) szerint then  
      hozzáad {var = érték} hozzárendelések-hez  
      eredmény  $\leftarrow$  REKURZÍV-VISSZALÉPÉSÉS(hozzárendelések, csp)  
      if eredmény  $\neq$  meghíúsulás then return eredmény  
      kivesz {var = érték} hozzárendelések-ből  
  return kudarc
```

†

1.2. ábra. Visszalépéses kereső algoritmus

2. fejezet

Témakörök, melyekre rálátással kell rendelkezni

2.1. Következtetések ítéletlogikában

51. Definíció. Vonzat. Azt mondjuk, hogy az α mondat maga után vonzza a β mondatot, akkor és csakis akkor, ha minden modellben, amelyben α igaz, β szintén igaz. Ezt a következőképp jelöljük:

$$\alpha \models \beta.$$

52. Definíció. Következtetési eljárás helyessége.

Egy következtetési eljárást, amely csak vonzat mondatokat vezet le, helyesnek vagy igazságtartónak nevezük. A helyesség egy igencsak kívánatos tulajdonság. Egy nem helyes következtetési eljárás kitalál olyan dolgokat ahogyan előrehalad, olyan tük megtalálását jelenti be, amelyek nem is léteznek.

53. Definíció. Következtetési eljárás teljessége. A teljesség tulajdonság szintén kívánatos: **egy következtetési eljárás teljes, ha képes levezetni minden vonzatmondatot.**

Valódi szénakazlak esetében, amelyek véges méretűek, nyilvánvalónak tűnik, hogy szisztematikus kutatással mindig eldönthető, hogy a tű a kazalban van-e. Sok tudásbázis esetében azonban a konzekvenciák szénakazlának mérete végtelen, és így a teljesség egy fontos kérdéssé válik.

2.1.1. Rezolúciókalkulus ítéletlogikában

54. Definíció. Konjunktív normálforma (KNF).

Egy mondatot, amelyet literálok diszjunkcióinak konjunkcióival fejezünk ki, **konjunktív normál formájúnak** (conjunctive normal form) vagy KNF formájúnak nevezünk.

Például:

$$(X_1 \vee X_2 \vee \dots X_n) \wedge (Y_1 \vee Y_2 \vee \dots Y_n).$$

55. Definíció. Rezolúció. A rezolúció egy levezetési eljárás, mely alapja az egyik automatikus tételbizonyítási módszernek, elméletnek, a rezolúciós kalkulusnak.

A rezolúció szintatikus módszer; alapja, hogy két logikai formulához hozzárendeljük egy speciális következményformulájukat, az ún. **rezolvensüket**.

56. Definíció. Zárt klóz. Véges sok negátatlan vagy negált zárt atomi formulából (literálból) diszjunkció által összetett formula egy **zárt klózt** alkot. Például

$$X \vee Y \vee Z.$$

Ennek speciális esete a **Horn-klóz**, amely olyan zárt klóz, mely legfeljebb egy negátatlan atomot tartalmaz, a többi tagja viszont negált. Például

$$A \vee \neg B \vee \neg C \vee \neg D.$$

9. Megjegyzés. A rezolúció felhasználható például SAT-problémák (kielégíthetőségi problémák) megoldására, azaz amikor egy formula KNF alakjáról el kell dönteni, hogy található-e hozzá olyan interpretáció, ahol a formula kielégíthető.

10. Megjegyzés. A rezolúció azért használatos, mert lényegesen gyorsabb, mint ha az ítéletlogikai szemantikai definíciók mentén számolnánk.

A rezolúción alapuló következtetési eljárások az ellentmondásokra vezető bizonyítások elvén működnek. Tehát annak megmutatásához, hogy $TB \models \alpha$, azt mutatjuk meg, hogy a $(TB \wedge \neg \alpha)$ kielégíthetetlen. Ezt az

ellentmondás bizonyításával végezzük el.

Algorithm 7: Egy egyszerű rezolúciós algoritmus ítéletkalkulushoz

```
1 Function Rezolválás( $TB, \alpha$ ):  
   input :  $TB$ , a tudásbázis, egy ítéletkalkulus mondat  
            $\alpha$ , a lekérdezés, egy ítéletkalkulus mondat  
2    $klózok \leftarrow$  a  $TB \wedge \neg\alpha$  mondatot reprezentáló CNF formulájú klózok halmaza  
3    $új \leftarrow \{\}$   
4   loop do  
5     foreach  $C_i, C_j$  in  $klózok$  do  
6        $rezolvensek \leftarrow$  Rezolválás( $C_i, C_j$ )  
7       if  $rezolvensek$  tartalmazzák az üres klózt then return igaz  
8        $új \leftarrow új \cup rezolvensek$   
9     end foreach  
10    if  $új \subseteq klózok$  then return hamis  
11  end loop
```

A rezolúció algoritmust mutatja a 7. algoritmus. Először a $(TB \wedge \neg\alpha)$ -t konvertáljuk CNF formára. Majd a rezolúciós szabályt alkalmazzuk a létrejövő klózokra. Minden egyes párt, amely kiegészítő literálokat tartalmaz, rezolválunk, hogy egy új klózt hozzunk létre, amelyet hozzáadunk a halmazhoz, ha még nem volt jelen. A folyamat addig folytatódik, amíg a következő két dolog közül valamelyik meg nem történik:

- nincs több új klóz, amit hozzá lehet adni, ilyen esetben α nem vonzza maga után β -t
- a rezolúció alkalmazása egy üres klózra vezet, amely esetben $\alpha \models \beta$.

2. Tétel. Az üres klóz – egy diszjunkt nélküli diszjunktció – ekvivalens a *Hamis* értékkel, mert a diszjunktció akkor igaz csak, ha legalább az egyik diszjunkt igaz. Vagyis beláthatjuk, hogy az üres klóz ellentmondást reprezentál, hogy megfigyeljük azt, hogy az üres klóz két kiegészítő egységklóz, mint amilyen az S és $\neg S$, rezolválásából származik.

2.2. Döntési fák

57. Definíció. Információs rendszer. Egy **információs rendszer** alatt egy $\mathcal{I} = \langle S, A, V, f \rangle$ négyest értünk, ahol

- S az objektumok halmaza,
- $A = \{a_1, a_2, \dots, a_n\}$ attribútumok véges nemüres halmaza, ahol minden $a \in A$ attribútumhoz tartozik egy nemüres V_a értékhalmaz (tartomány)
- V az attribútumértékek halmaza: $V = \bigcup_{i=1}^k V_{a_i}$
- $f : S \times A \rightarrow V$ függvény úgy, hogy

$$f(s, a) \in V_a \text{ minden } s \in S \text{ és } a \in A \text{ esetén.}$$

58. Definíció. Döntési tábla. **Döntési tábla** alatt olyan $\langle \mathcal{I}, d \rangle$ rendezett párt értünk, ahol

- $\mathcal{I} = \langle S, A, V, f \rangle$ egy információs rendszer,
- $d \in A$ a **döntési attribútum**.

Ekkor az attribútumok $C = A \setminus \{d\}$ részhalmaza a **feltétel attribútumok** (condition attributes) halmaza.

A döntési fa tanulás egyike a legegyszerűbb, mégis az eddigiekben az egyik legsikeresebbnek bizonyult tanulási algoritmusnak. Jó bevezetésként szolgál az induktív tanulás területén, és ráadásul könnyen implementálható. Először bemutatjuk a cselekvő alrendszert, majd megmutatjuk, hogy miképpen lehet tanítani. Mindeközben olyan elveket mutatunk be, amelyek az induktív tanulás minden területére jellemzők.

59. Definíció. Induktív tanulás. Cél, hogy pozitív és negatív példák alapján olyan logikai függvényt állítsunk elő, ami igaz értéket ad a pozitív és hamisat a negatív példákra és feltételezzük, hogy ha elég nagyszámú példánk van akkor helyes értéket fogunk kapni előre nem látott példák esetén is.

60. Definíció. Döntési fa.

Egy döntési fa (decision tree) bemenetként egy attribútumokkal (attributes) leírt objektumot vagy szituációt kap, és egy „döntést” ad vissza eredményként – a bemenetre adott válasz jóslott értékét. A bemeneti attribútumok lehetnek diszkrét vagy folytonosak. Jelen tárgyalásban diszkrét bemeneteket tételezünk fel. A kimeneti érték szintén lehet diszkrét vagy folytonos; egy diszkrét értékkészletű függvény tanulását osztályozás (classification) tanulásnak, míg a folytonos függvény tanulását regresszióknak (regression) nevezzük. Bináris (Boolean) osztályozásra fogunk koncentrálni, ahol minden példát vagy igaznak (pozitív), vagy hamisnak (negatív) sorolunk be.

2.2.1. Az ID3 algoritmus

Algorithm 8: ID3

```

1 Function ID3(  $\mathcal{I}, d, S', C', b$ ):
2   if  $S' = \emptyset$  then
3     | return new-node-with-label( $b$ )
4   end if
5    $m \leftarrow \text{majority}(S', d)$ 
6   if  $A = \emptyset$  or  $\forall s \in S' : (f(s, d) = m)$  then
7     | return new-node-with-label( $m$ )
8   end if
9    $a \leftarrow \text{select}(A)$ 
10   $C'' \leftarrow C' \setminus \{a\}$ 
11   $r \leftarrow \text{new-node-with-label}(a)$ 
12  forall  $v \in V_a$  do
13    |  $S'' \leftarrow \{u : u \in S' \wedge f(u, a) = v\}$ 
14    |  $t \leftarrow \text{ID3}(\mathcal{I}, d, S'', C'', m)$ 
15    | add-edge( $r, t, v$ )
16  end forall
17  return  $r$ 

```

2.3. Valószínűségi következtetés

Bizonytalanság

Jelölje A_t azt a műveletet, hogy t perccel a vonat indulása előtt elindulunk a pályaudvarra. A_t -vel időben odaérünk? A következő problémák lépnek fel:

- részlegesen megfigyelhető környezet (utak állapota, más sofőrök terve, stb.)
- zajos észlelések (Útinfo, DKV applikáció jelzései)
- a művelet kimenelének bizonytalansága (lapos gumi, stb.)
- a közlekedés modellezésének és előrejelzésének hatalmas bonyolultsága

Logikai megközelítésben A probléma logikai megközelítése:

- kockázat elutasítása: " A_{25} esetén odaérek"
- pontos megfogalmazás (túl határozatlan döntéshozatalhoz) " A_{25} -tel odaérek, ha nincs útközben baleset, útfelújítás, és nem történik addig a kocsival semmi, stb."

A_{120} biztosan elég (még gyalog is odaértek), de senki nem akar ennyi időt az állomáson tölteni.

Bizonytalanság kezelésének módszerei

- Alapértelmezett vagy nem-monoton logika
 - feltesszük, hogy a kocsinak nem lyukad ki a gumija
 - feltesszük, hogy A_{25} elég, hacsak nem mond ellent a nyilvánvaló tényeknek
 - Mely feltételek ésszerűek? Hogyan kezelhetjük az ellentmondásokat?
- Szabályok tapasztalati tényezőkkel
 - $A_{25} \Rightarrow_{0.3}$ Időben kiér az állomásra
 - locsol $\Rightarrow_{0.99}$ vizes a fű
 - vizes a fű $\Rightarrow_{0.7}$ esett az eső
 - szabályok kombinálása: A locsol következménye az esett az eső?
- Valószínűség
 - A nyilvánvaló tények mellett A_{25} esetén 4% eséllyel jutok el időben az állomásra.
- Fuzzy logika
 - igazság foka (és nem bizonytalanság)
 - "a fű vizes" értéke 0.2
 - * nem "100 esetben 20-szor vizes a fű"
 - * hanem "enyhén vizes a fű"

Valószínűség

Valószínűségi feltételek összegzik a

- lustaságot: túl sok munkát jelent az ok és okozat teljes eseményhalmazának felsorolása
- elméleti tudatlanságot: a tudományterület elmélete nem teljes
- gyakorlati tudatlanságot: nem végezhető el az összes vizsgálat

Észlelések alkotják a tény/tényállást

- a kihúzott lap pikk ász ($1/52$ valószínűséggel - lap megnézése nélkül)
 - megnézéssel 0 vagy 1 valószínűséggel
- mielőtt a tények birtokába jutunk előzetes/a priori valószínűség
- tények birtokában utólagos/a posteriori/feltételes valószínűség

A valószínűségek személyes meggyőződésekhez/hiedelmekhez kapcsolódhatnak.

Döntések bizonytalanság mellett

Tekintsük a következő feltételezéseket:

- $P(A_{25}\text{-tel időben odaérek}|\dots) = 0.04$
- $P(A_{30}\text{-cal időben odaérek}|\dots) = 0.23$
- $P(A_{75}\text{-tel időben odaérek}|\dots) = 0.74$
- $P(A_{120}\text{-szal időben odaérek}|\dots) = 0.999$

Ekkor az, hogy melyik műveletet választjuk függ a személyes preferenciáktól: lekészt vonat, várakozás, ...

- Hasznosságelmélet: preferenciák ábrázolása és felhasználása
- Döntéelmélet = hasznosságelmélet + valószínűség

Valószínűségszámítás alapjai

61. Definíció. Elemi események, elemi események halmaza. Legyen Ω az elemi események egy halmaza (pl. a kockadobás hat kimenetele). Ekkor $\omega \in \Omega$ egy elemi esemény, egy lehetséges világ. Az elemi események kölcsönösen kizárják egymást.

Minden elemi eseményhez rendelhető egy valószínűség:

$$0 \leq P(\omega) \leq 1,$$

ahol $\sum P(\omega) = 1$.

3. Tétel. Ha egy A esemény adott elemei események uniója, akkor az A valószínűsége az adott elemi események valószínűségének összege.

62. Definíció. A valószínűségi változó egy függvény, mely az elemi eseményekhez egy értéket (rendszerint valós, vagy logikai) rendel, például páratlan(1) = igaz, vagy páros(3) = hamis.

A P bármely X valószínűségi változóhoz valószínűségi eloszlást rendel:

$$P(X = x_i) = \sum_{\omega: X(\omega) = x_i} P(\omega).$$

A valószínűségeket azért használjuk, mert a logikailag összekapcsolódó események valószínűségei is összekapcsolódnak:

$$P(a \vee b) = P(a) + P(b) - P(a \wedge b).$$

Állítások

Állításnak azt tekintjük, amikor egy esemény (elemi események halmaza) teljesül.

Legyen A és B két véletlen változó. Ekkor

- a esemény = azon elemi események halmaza, ahol $A(\omega) = \text{igaz}$
- $\neg a$ esemény = azon elemi események halmaza, ahol $A(\omega) = \text{hamis}$
- $a \wedge b$ esemény = azon elemi események halmaza, ahol $A(\omega) = \text{igaz}$ és $B(\omega) = \text{igaz}$

MI alkalmazásokban az elemi eseményeket a valószínűségi változók értékei határozzák meg. Pl. az elemi események halmaza az értékészletek Descartes-szorzata.

Logikai változók esetén az elemi esemény egy nulladrendű interpretáció.

Egy állítás elemi állítások diszjunkciója.

4. Tétel. Állítások szintaxisa.

- Ítéletlogikai vagy logikai valószínűségi változó
 - Lyuk (lyukas a fogam?)
 - "Lyuk = igaz" egy állítás, leírható lyukas alakban is
- Diszkrét valószínűségi változó (véges vagy végtelen)
 - az Időjárás a (napos, esős, felhős és havas) egyike
 - "Időjárás = esős" egy állítás
 - az értékeknek egymást kizárónak kell lenniük, és ki kell adniuk az összes lehetőséget
- Folytonos valószínűségi változó (korlátos vagy nem korlátos)
 - "Hőmérséklet = 21.6" vagy "Hőmérséklet < 22.1" állítások

5. Tétel. Állítások bármilyen logikai kombinációja újabb állítást ad.

Priori – előzetes valószínűség

A priori állítások előzetes, vagy feltétel nélküli valószínűsége. Azt a meggyőződési mértéket jelenti, amely bármely más információ hiányában az állításhoz kapcsolható. Pl.:

$$P(\text{lyukas}) = 0.1, \quad P(\text{Időjárás}=\text{napos}) = 0.73.$$

A valószínűségi eloszlás értéket rendel minden lehetséges értékadáshoz. Pl.:

$$P(\text{Időjárás}) = (0.73; 0.1; 0.07; 0.1).$$

Ez a mennyiség normalizált, az egyes eshetőségek összege 1.

Együttes valószínűségi eloszlás: véletlen változóhalmaz összes lehetséges kombinációjának valószínűsége.

$P(\text{Időjárás}, \text{Lyuk}) =$	Időjárás	napos	esős	felhős	havas
	Lyuk = igaz	0.144	0.02	0.016	0.02
	Lyuk = hamis	0.576	0.08	0.064	0.08

Folytonos változók valószínűsége

A lehetséges értékek száma végtelen, így táblázatban nem foglalható össze. Annak valószínűsége, hogy egy valószínűségi változó egy adott x értéket vesz fel, általában x egy paraméterezett függvényeként definiálható. Pl.

$$P(\text{Hőmérséklet} = x) = \bigcup_{i=18}^{26} (x)$$

– azaz egyenletes eloszlású 18 és 26 °C között.

$P(X = 20.5) = 0.125$ értelmezése:

$$\lim_{dx \rightarrow 0} \frac{P(20.5 \leq X \leq 20.5 + dx)}{dx} = 0.125.$$

2.3.1. Bayes hálók, Bayes tétel

6. Tétel. A Bayes-tétel:

$$P(b|a) = \frac{P(a|b)P(b)}{P(b)}.$$

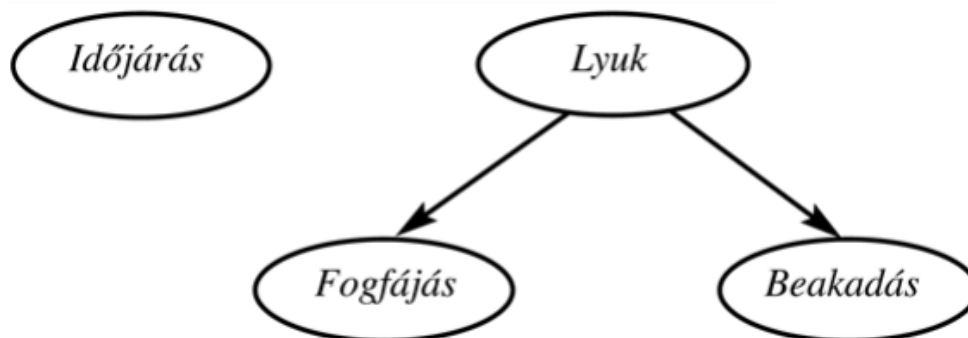
7. Tétel. A Bayes-tétel általános eloszlásokra

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)}.$$

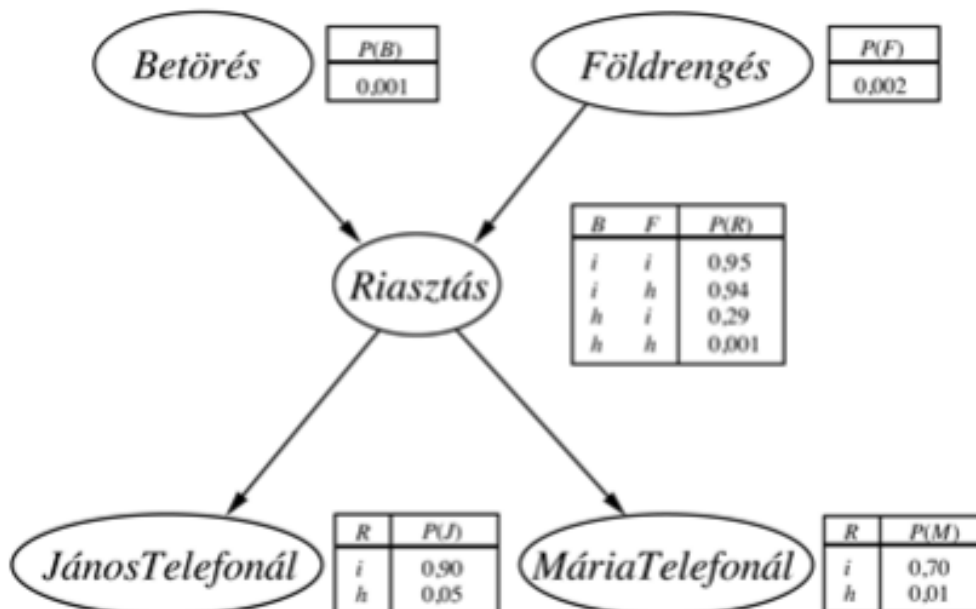
63. Definíció. A Bayes-hálók egy a bizonytalanság esetén használható tudás-reprezentációs módszer adatstruktúrája. A Bayes-hálók alkalmasak változók közötti függőség leírásához, és bármely együttes valószínűség-eloszlás függvény tömör megadásához.

A Bayes-háló egy irányított gráf, amelyben minden csomóponthoz számszerű valószínűségi információk vannak csatolva. A teljes megadás a következő:

1. A háló csomópontjait valószínűségi változók egy halmaza alkotja. A változók lehetnek diszkrét, vagy folytonosak.
2. Irányított élek (nyilak) egy halmaza összeköt bizonyos csomópontpárokat. Ha létezik nyíl az X csomóponttól az Y csomópontig, azt mondjuk, X szülője az Y -nak.
3. Minden X_i csomóponthoz tartozik egy $P(X_i | \text{Szülők}(X_i))$ feltételes valószínűségi eloszlás, ami számszerűen megadja a szülők hatását a csomóponti változóra.
4. A gráf nem tartalmaz irányított kört (azaz irányított, körmentes gráf).



2.1. ábra. Egy egyszerű Bayes-háló, amelyben az Időjárás független a többi három változótól, a Fogfájás és Beakadás pedig feltételesen függetlenek a Lyuk ismeretében



2.2. ábra. Egy tipikus Bayes-háló, amely a topológiát és a feltételes valószínűségi táblákat (FVT) is mutatja.

2.3.2. Feltételes valószínűség számítása

Feltételes valószínűség

Ha az ágens bizonyos tények birtokába jut a korábban ismeretlen, a tartományra jellemző véletlen változóra vonatkozóan, akkor a priori valószínűségek nem használhatóak. Például

$$P(\text{lyukas}|\text{fogfájás}) = 0.8$$

esetén

- adott, hogy a betegnek fáj a foga; ezt és csak ezt tudjuk
- nem arról van szó, hogy "ha fáj a foga, akkor 80% eséllyel lyukas"
- hanem "ha fáj a foga és semmilyen más információnk nincs, akkor 80% eséllyel lyukas"

Képlet

A feltételes valószínűség definíciója (ha $P(b) \neq 0$)

$$P(a|b) = \frac{P(a \wedge b)}{P(b)}.$$

Szorzat szabály

A szorzat szabály egy alternatív megfogalmazás:

$$P(a \wedge b) = P(a|b)P(b) = P(b|a)P(a).$$

Láncszabály

A szorzatszabály többszöri alkalmazásával kapjuk meg a láncszabályt:

$$P(X_1, \dots, X_n) = P(X_n|X_1, \dots, X_{n-1}) \cdot P(X_1|X_1, \dots, X_{n-1}) = \prod_{i=1}^n P(X_i|X_1, \dots, X_{i-1}).$$

Feltételes valószínűség kiszámítása

$$\begin{aligned} P(\text{lyuk}|\text{fogfájás}) &= \frac{P(\text{lyuk}) \wedge \text{fogfájás}}{P(\text{fogfájás})} = \frac{0.108 + 0.012}{0.018 + 0.012 + 0.016 + 0.064} = 0.6 \\ P(\neg\text{lyuk}|\text{fogfájás}) &= \frac{P(\neg\text{lyuk}) \wedge \text{fogfájás}}{P(\text{fogfájás})} = \frac{0.016 + 0.064}{0.018 + 0.012 + 0.016 + 0.064} = 0.4 \end{aligned}$$

2.4. Neurális hálók, kitekintés

Néhány kihívás és néhány megoldás mesterséges intelligencia segítségével

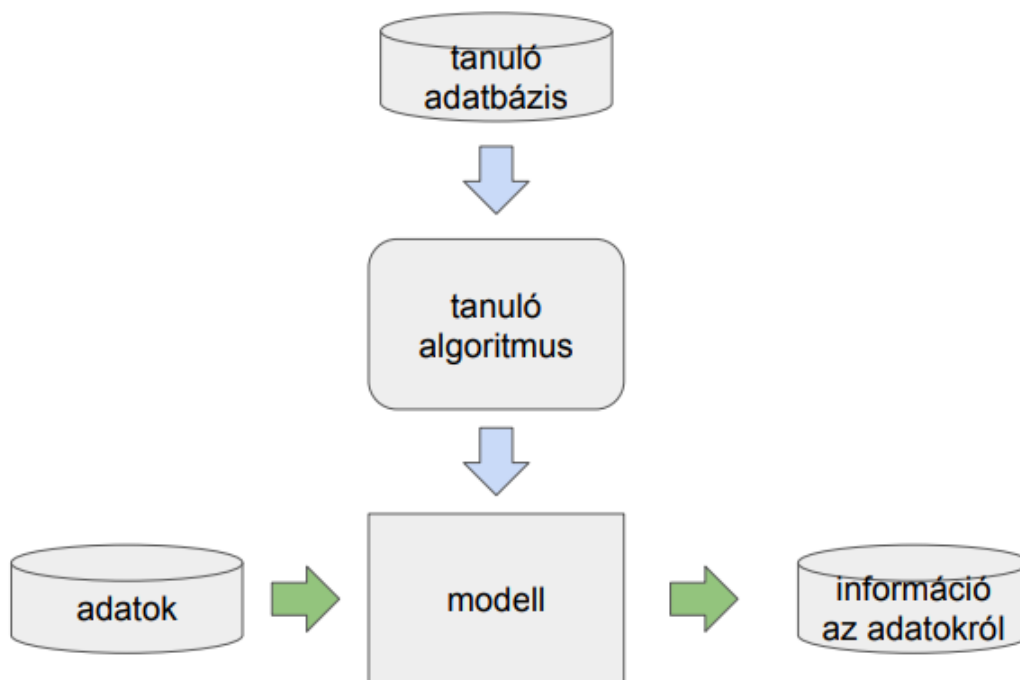
- nagy tömegű adat tárolása, visszakeresése
- beszéd előállítás
 - Pocket - demo
- természetes nyelv megértése
 - Siri
 - Google Assistant
 - Cortana
 - Alexa
- gépi fordítása
 - Google Translate
- alakfelismerés
 - Face Unlock
- robotvezérlés
 - Boston Dynamics
- zeneszerzés
- zenefelismerés
 - Shazam

Példák gépi tanulásra

- spamszűrés
- karakterfelismerés
- fotók címkézése
- ajánló rendszerek
- szociális háló elemzése
- hírek csoportosítása témájuk alapján

Alapfogalmak

- **Tanuló adatbázis:** párok halmaza (pl. email-címke)
- **Modell:** tudás ami alapján címkézhetünk
- **Adatok:** e-mailek
- **Információ az adatokról:** spam/nem spam címkékk



2.3. ábra. Gépi tanulás alapfogalmai

Felügyelt gépi tanulás esetén rendelkezésre állnak a közvetlen információk. Pl.: spam/nem spam, karakterek, stb. (osztályozási feladatok).

Felügyelet nélküli gépi tanulás esetén nincs közvetlen információ. Pl.: csoportosítás/klaszterezés.

64. Definíció. Regresszió.

Az eredményváltozó hogyan függ a más (magyarázó) változók alakulásától.

- lineáris: $\hat{y} = a + bx$
 - legkisebb négyzetek módszere
- nem lineáris: $y = Ae^{Bx}$
- több osztályú problémák

65. Definíció. Osztályozás.

Modell szeparáló egyenesek (hipersíkok).

Módszerek:

- Bayes osztályozók
- Döntési fák
- Support Vector Machines

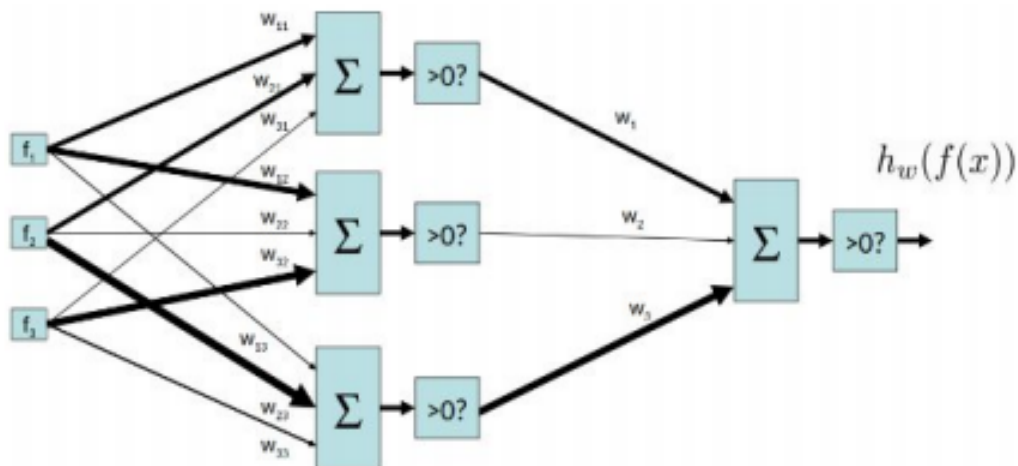
66. Definíció. Klaszterezés.

- Csoportok keresése - felügyelet nélküli tanulása
- Exkluzív (k-közép) vagy átfedő (fuzzy) módszer
- főkomponens analízis (PCA)
- hierarchikus

2.4.1. Neurális hálók és mélytanulás

Mesterséges neuronhálózat

67. Definíció. Neuron: számítási egység.



2.4. ábra. Neuronhálózat

- több x_i bemenet (mindegyikhez w_i súly rendelve)
- egy kimenet
 - izgatottság: $\sum x_i w_i$
 - ha az izgatottság túllép egy határszintet, akkor a neuron tüzel

8. Tétel. Egy kétszintű neuronhálózat elegendő számú neutronnal bármely folytonos függvényt képes megközelíteni tetszőleges pontossággal.

9. Tétel. Mesterséges neuronhálózat szerkezete:

- teljesen kapcsolt réteg:

$$H = XW + b$$

- egyszerű konkurens réteg:

$$H_t = XW_x + b_x + H_{t-1}W_h + b_h$$

- konvolúciós réteg:

$$H = X * W + b.$$

- Aktivációs réteg:

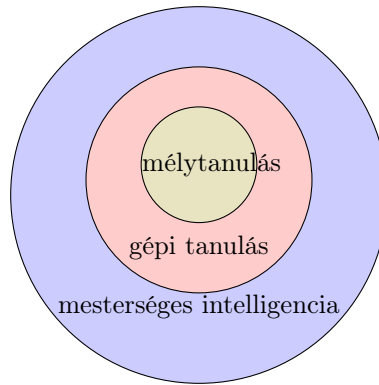
$$H = g(X).$$

Egy neuronhálózat tanítása számolásigényes feladat, a tanítás akár több hét is lehet, az alkalmazás viszont gyors.

Tanítási módszerek:

- hiba-visszaterjesztés
 - az elvárt és kapott kimenet különbségénél megbecsüljük az egyes súlyok hozzájárulását a hibához
 - * iránymenti deriváltak meghatározása, gradiens irányban elmozdulás
 - az egyes súlyokat korrigáljuk ennek megfelelően (se túl kicsit, se túl sokat)
 - zajos adatokkal nehezen boldogul
- túltanulás elleni védekezés
 - ritkítjuk a súlymátrixot (több 0 érték)
 - neuronok egy halmazát kirejti a hálózatból

Mélytanulás



2.5. ábra. A mélytanulás és a mesterséges intelligencia kapcsolata

A mélytanulást a következők teszik szükségessé:

- Bonyolultabb feladatoknál a hagyományos módszerek rosszul teljesítenek (a jellemzők kinyerése nehéz munka).
- Amennyiben az input nagyobb méretű, a "keskeny" neuronhálózatok nem elegendőek
 - a neuronok száma exponenciális növelendő
 - a pontosság csapnivaló, a tanulás lassú

Irodalomjegyzék

- [1] **Russell** Stuart J, **Norvig** Peter: Mesterséges Intelligencia modern megközelítésben. Panem Kft., 2005.
- [2] Várterész Magda: A mesterséges intelligencia alapjai: Az előadások mellé vetített anyag (2011).