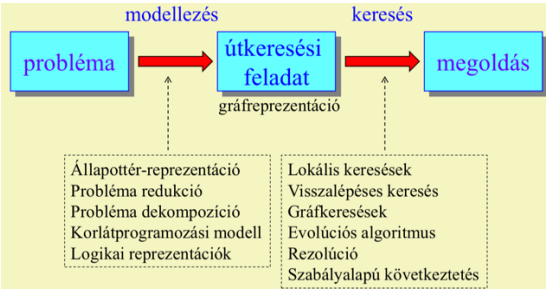
Mesterséges Intelligencia

1. Miről ismerhető fel az MI?
   – Megoldandó feladatai: nehezek. A feladat problématere hatalmas, a megoldás megkeresése kellő intuíció hiányában kombinatorikus robbanáshoz vezethet.
   – A megoldást biztosító szoftverek intelligensen működnek. Úgy, mintha a háttérben egy ember tevékenykedne (Turing teszt), továbbá gyakran tanulnak működésük közben, és ennek következtében javulhat az eredményességük és hatékonyságuk.
   – Megoldási módszereire az átgondolt reprezentáció és heurisztikával megerősített algoritmusok használata jellemző

2. Modellezés és keresés



3. Útkeresési probléma
   A problématerének elemeit egy speciális élsúlyozottirányított gráf csúcsai vagy útjai szimbolizálják.
   A megoldást ennek megfelelően vagy egy célcsúcs, vagy egy startcsúcsból célcsúcsba vezető (esetleg a legolcsóbb ilyen) út megtalálása szolgáltatja.
4. Gráf fogalmak

- csúcsok, irányított élek N, $A \subseteq N \times N$ (végtelen számosság)
- él n-ből m-be $(n,m) \in A$ $(n,m \in N)$
- n utódai $\Gamma(n) = \{m \in N \mid (n,m) \in A\}$
- n szülei $\pi(n) \in \Pi(n) = \{m \in N \mid (m,n) \in A\}$
- irányított gráf $R=(N,A)$
- véges sok kivezető él $|\Gamma(n)| < \infty \quad (\forall n \in N)$
- élköltség $c: A \to \mathbb{R}$
- δ-tulajdonság ($\delta \in \mathbb{R}^+$) $c(n,m) \geq \delta > 0 \quad (\forall (n,m) \in A)$
- δ-gráf δ-tulajdonságú, véges sok kivezető élű, élsúlyozott irányított gráf

- irányított út $\alpha = (n,n_1),(n_1,n_2),\ldots,(n_{k-1},m) = \langle n,n_1,n_2,\ldots,n_{k-1},m \rangle$
  ráfokban ez végtelen sok setén is értelmes. ke ∞, ha nincs egy út se.
  $n \to^{\alpha} m$, $n \to m$, $n \to M$, $\{n \to m\}$, $\{n \to M\}$ $(M \subseteq N)$
- út hossza az út éleinek száma: $|\alpha|$
- út költsége $c(\alpha) = c^{\alpha}(n,m) := \Sigma_i\, c(n_{i-1},n_i)$ ha $\alpha = \langle n=n_0,n_1,n_2,\ldots,n_{k-1},m=n_k \rangle$
- opt. költség $c^*(n,m) := \min_{\alpha \in \{n \to m\}} c^{\alpha}(n,m)$, $c^*(n,M) := \min_{\alpha \in \{n \to M\}} c^{\alpha}(n,m)$
- opt. költségű út $n \to^* m := \min_c \{\alpha \mid \alpha \in \{n \to m\}\}$, $n \to^* M := \min_c \{\alpha \mid \alpha \in \{n \to M\}\}$

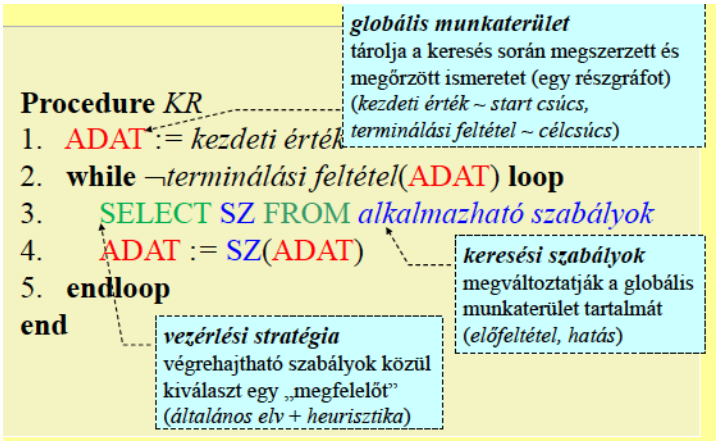
5. Gráfreprezentáció fogalma

- Minden útkeresési probléma rendelkezik egy (a probléma modellezéséből származó) gráfreprezentációval, ami egy (R,s,T) hármas, amelyben
  - $R=(N,A,c)$ δ-gráf az ún. reprezentációs gráf,
  - az $s \in N$ startcsúcs,
  - a $T \subseteq N$ halmazbeli célcsúcsok.
- és a probléma megoldása:
  - egy $t \in T$ cél megtalálása, vagy
  - egy $s \to T$, esetleg $s \to^* T$ optimális út megtalálása

6. Keresés

- Az útkeresési problémák megoldásához a reprezentációs gráfjaik nagy mérete miatt speciális (nem determinisztikus, heurisztikus) útkereső algoritmusokra van szükség, amelyek
  - a startcsúcsból indulnak, amely az első aktuális csúcs;
  - minden lépésben nem-determinisztikus módon új aktuális csúcsot választanak a korábbi aktuális csúcs(ok) gyerekei közül;
  - tárolják a már feltárt reprezentációs gráf egy részét;
  - megállnak, ha célcsúcsot találnak vagy nyilvánvalóvá válik, hogy erre semmi esélyük.

7. Kereső rendszer



8. Kereső rendszerek vizsgálata
   - helyes-e
   - teljes-e
   - optimális-e
   - idő bonyolultág
   - tár bonyolultság
9. Állapottér-reprezentáció

   Állapottér: a probléma leírásához szükséges adatok által felvett érték-együttesek (azaz állapotok) halmaza – az állapot többnyire egy összetett szerkezetű érték – gyakran egy bővebb alaphalmazzal és egy azon értelmezett invariáns állítással definiáljuk

   Műveletek: állapotból állapotba vezetnek – megadásukhoz: előfeltétel és hatás leírása – invariáns tulajdonságot tartó leképezés

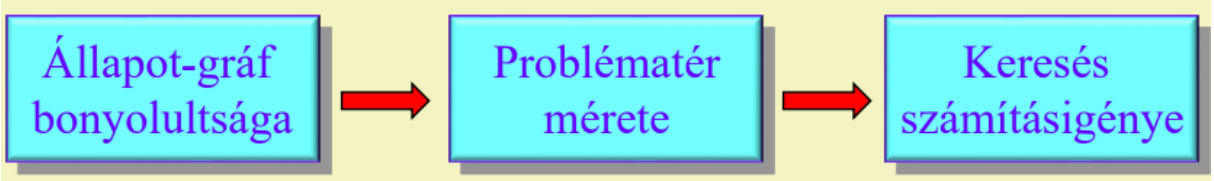
   Kezdőállapot(ok) vagy azokat leíró kezdeti feltétel

   Végállapot(ok) vagy célfeltétel
10. Állapottér-reprezentáció gráf-reprezentációja

    Állapot-gráf állapot — csúcs
    - művelet hatása — irányított él
    - művelet költsége — élköltség

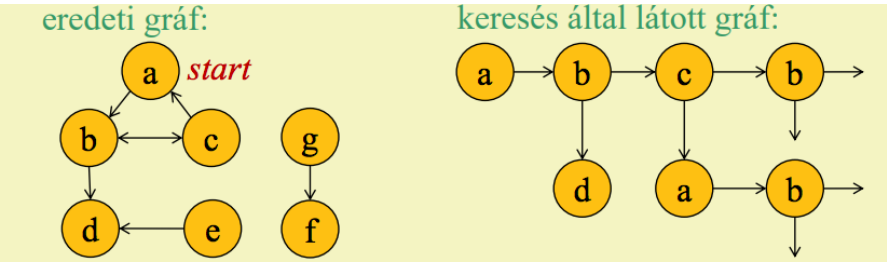
    Reprezentációs gráf δ-gráf — állapot-gráf
    - startcsúcs — kezdőállapot
    - célcsúcsok — végállapotok
    - irányított út — egy műveletsorozat hatása
    - irányított út a startból a célba — egy megoldás

11. Állapottér vs. problématér
A problématér elemeit többnyire a start csúcsból kiinduló utak szimbolizálják.
Az állapottér-reprezentáció és a problématér között szoros kapcsolat áll fenn, de az állapottér többnyire nem azonos a problématérrel.
12. Állapot-gráf bonyolultsága



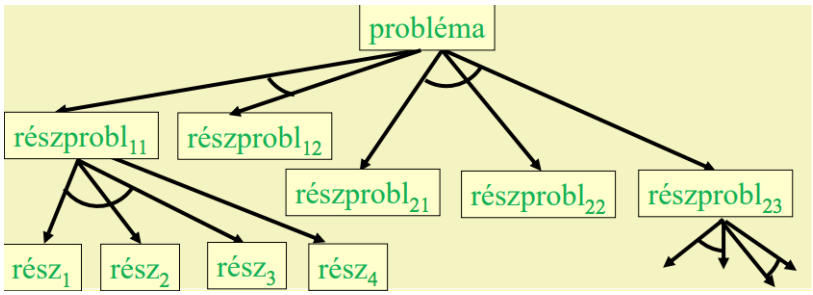
- A start csúcsból kivezető utak száma az oda-vissza lépések nélkül
  Hanoi: a legfeljebb k hosszú utak: $1+2+\ldots+2^k$, azaz $2^{k+1}-1$
  - csúcsok és élek száma
    Hanoi: 3^n csúcs, $3 \cdot \frac{3^n - 1}{3 - 1}$ él
  - a csúcsok átlagos ki-foka
    Hanoi: 3
  - körök gyakorisága, és hosszuk sokfélesége
    Hanoi: 2, 3, 6, 7, 9, …

13. Hogyan „látja” egy keresés a reprezentációs gráfot?
Egy keresés fokozatosan fedezi fel a reprezentációs gráfot: bizonyos részeihez el sem jut és a felfedezett részt sem feltétlenül tárolja el teljesen. Sőt kifejezetten torzultan „látja” a gráfot, ha egy csúcshoz érve nem vizsgálja meg, hogy ezt a csúcsot korábban már felfedezte-e, hanem ehelyett új csúcsként regisztrálja.



14. Reprezentációs gráf „fává egyenesítése”
Ha a keresés nem vizsgálja meg egy csúcsról, hogy korábban már felfedezte-e, akkor az eredeti reprezentációs gráf helyett valójában annak fává kiegyenesített változatában keres.
Előny: eltűnnek a körök, de a megoldási utak megmaradnak
Hátrány: duplikátumok jelennek meg, sőt a körök kiegyenesítése végtelen hosszú utakat eredményez
A kétirányú (oda-vissza) élek különösen megnövelik a kiegyenesített fa méretét. Ezek kiszűrése azonban könnyen beépíthető minden keresésbe a megelőző aktuális csúcs (szülőcsúcs) eltárolásával, így az aktuális csúcsból oda visszavezető él felismerhető és figyelmen kívül hagyható.
15. Probléma dekompozíció
- Egy probléma dekomponálása során a problémát részproblémákra bontjuk, majd azokat tovább részletezzük, amíg nyilvánvalóan megoldható problémákat nem kapunk.

- Sokszor egy probléma megoldását akár többféleképpen is fel lehet bontani részproblémák megoldásaira.

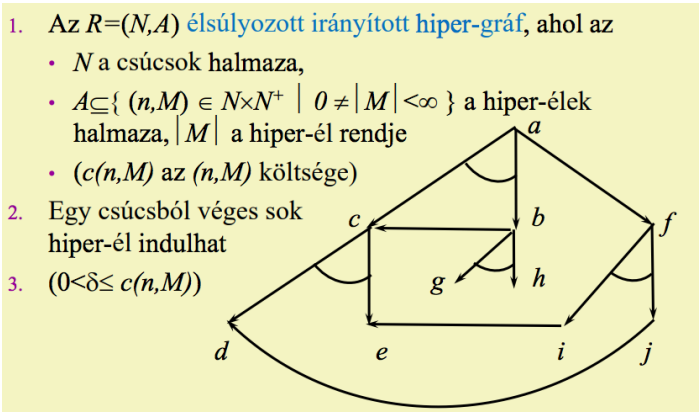


16. Dekompozíciós reprezentáció fogalma
A reprezentációhoz meg kell adnunk:
– a feladat részproblémáinak általános leírását,
– a kiinduló problémát,
– az egyszerű problémákat, amelyekről könnyen eldönthető, hogy megoldhatók-e vagy sem, és
– a dekomponáló műveleteket:
  • D: probléma → probléma$^+$ és D(p)=‹p_1, ... , p_n›

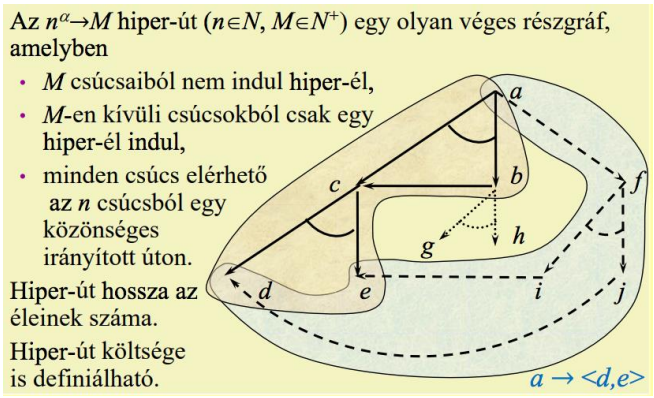
17. A dekompozíció modellezése ÉS/VAGY gráffal
<u>Egy dekompozíciót egy ÉS/VAGY gráffal szemléltethetünk:</u>
– a csúcsok részproblémákat jelölnek, a kiinduló problémát a startcsúcs, a megoldható egyszerű problémákat a célcsúcsok.
– egy dekomponáló művelet hatását egy élköteg mutatja, amely a dekomponált probléma csúcsából a dekomponálással előállított részproblémák csúcsaiba vezet.

18. Megoldás-gráf
Az eredeti problémát egyszerű problémákra visszavezető dekomponálási folyamatot az ÉS/VAGY gráf speciális részgráfja, az ún. megoldás-gráf jeleníti meg, amelyben • -
- út vezet a startcsúcsból minden csúcsba, és minden csúcsból egy célcsúcsba
- egy éllel együtt az összes azzal „ÉS” kapcsolatban álló éleket is tartalmazza (azaz teljes élkötegeket tartalmaz)
- nem tartalmaz „VAGY” kapcsolatban álló él párokat

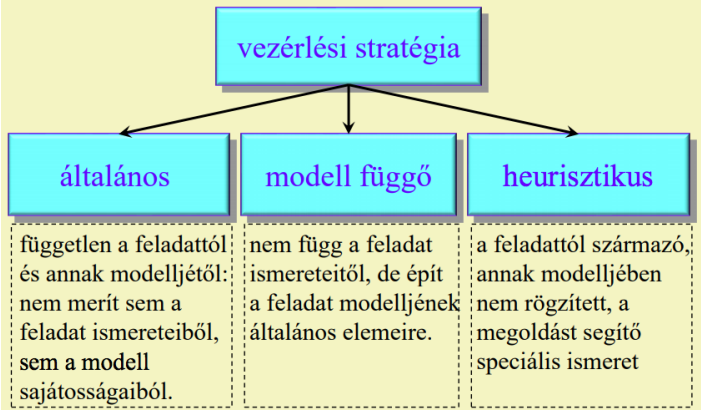
19. ÉS/VAGY gráfok



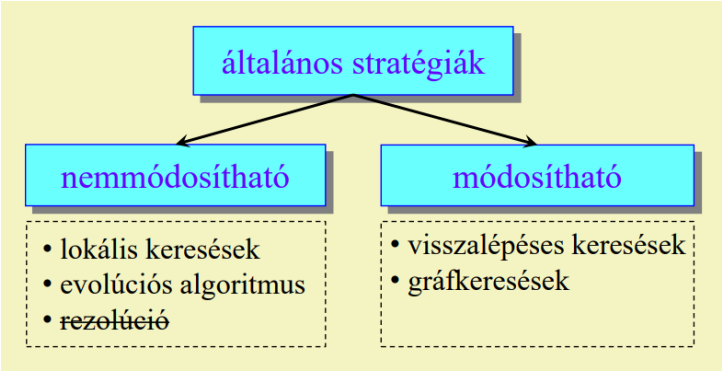
20. Az n csúcsból az M csúcs-sorozatba vezető irányított hiper-út fogalma



21. A hiper-út és a megoldás kapcsolata
-Az $n^{\alpha}\rightarrow M$ hiper-út egy egyértelmű haladási irányt jelöl ki az n csúcsból az M csúcsaiba.
- A probléma dekompozíciónál említett megoldás-gráf nem más, mint egy olyan hiper-út, amely a startcsúcsból egy célcsúcs-sorozatba vezet. Ez a célcsúcs-sorozat adja az eredeti probléma megoldását.
-A megoldás-gráf megtalálásához tehát a startcsúcsból kivezető hiper-utakat kell megvizsgálni.
-A startcsúcsból kivezető hiper-utak szerepe ugyanaz, mint a közönséges irányított gráfokban a startcsúcsból induló közönséges irányított utaké.
22. Különbség a közönséges út és a hiper-út bejárása között
-Egy közönséges irányított út bejárásán az úton fekvő csúcsoknak az élek által mutatott sorrendjében történő felsorolását értjük. Ez mindig egyértelmű.
-Egy irányított hiper-út bejárása is egy sorozat, de ez csúcs-sorozatok sorozata, ami nem egyértelmű.
23. Hiper-út bejárása
Az $n\rightarrow M$ hiper-út egy bejárásán a hiper-út csúcsaiból képzett sorozatoknak a felsorolását értjük:
• első sorozat:
• a C sorozatot a C $k\leftarrow K$(felső index) sorozat követi (ahol C-ben k minden előfordulásának helyére a K sorozatot írjuk), ha a hiper- útnak van olyan (k,K) hiper-éle, ahol $k\in C$ de $k\notin M$.
Megjegyzés:
– Egy hiperútnak véges sok véges hosszú bejárása van.
– Egy $n\rightarrow T$ hiper-út (azaz egy megoldás-gráf) bejárásainak utolsó eleme egy csupa célcsúcsot tartalmazó sorozat.
24. Útkeresés ÉS/VAGY gráfban
Egy ÉS/VAGY gráf startból induló hiper-útjainak (a potenciális megoldás gráfoknak) a bejárásai közönséges irányított utakként ábrázolhatók, és ezáltal egy közönséges irányított gráfot írnak le, amelynek csúcsai az eredeti ÉS/VAGY gráf csúcsainak véges sorozatai.
Ha ebben a közönséges gráfban megoldási (azaz csupa célcsúcsot tartalmazó sorozatba vezető) utat találunk, akkor az egyben az eredeti ÉS/VAGY gráf megoldás-gráfja is lesz.
Az ÉS/VAGY gráfbeli megoldás-gráf keresést tehát közönséges irányított gráfbeli keresést végző útkereső algoritmusokkal végezhetjük el.
25. KR vezérlési szintjei



26. Általános vezérlési stratégiák



27. Lokális keresések

Egyetlen aktuális csúcsot és annak szűk környezetét tárolják a globális munkaterületen. Keresési szabályai az aktuális csúcsot minden lépésben a szomszédjai közül vett lehetőleg „jobb” gyerekcsúccsal cserélik le. A vezérlési stratégiájuk a „jobbság” eldöntéséhez egy rátermettségi függvényt használ, amely annál jobb értéket ad egy csúcsra, minél közelebb esik az a célhoz. Mivel a keresés „elfelejti”, hogy honnan jött, a döntések nem vonhatók vissza, ez egy nemmódosítható vezérlési stratégia.
Lokális kereséssel megoldható feladatok azok, ahol egy lokálisan hozott rossz döntés nem zárja ki a cél megtalálását. Ehhez vagy egy erősen összefüggő reprezentációs-gráf, vagy jó heurisztikára épített célfüggvény kell. Jellemző alkalmazás: adott tulajdonságú elem keresése, függvény optimumának keresése.

• **Hegymászó algoritmus**: Minden lépésben az aktuális csúcs legjobb gyermekére lép, de kizárja a szülőre való visszalépést. Zsákutcába (aktuális csúcsból nem vezet ki él) beragad, körök mentén végtelen ciklusba kerülhet, ha a rátermettségi függvény nem tökéletes.

• **Tabu keresés**: Az aktuális csúcson (n) kívül nyilvántartja még az eddig legjobbnak bizonyult csúcsot (n *) és az utolsó néhány érintett csúcsot; ez a (sor tulajdonságú) tabu halmaz. Minden lépésben az aktuális csúcs gyermekei közül, kivéve a tabu halmazban levőket, a legjobbat választja új aktuális csúcsnak, (ezáltal felismeri a tabu halmaz méreténél nem nagyobb köröket), frissíti a tabu halmazt, és ha n jobb, mint az n * , akkor n * -ot lecseréli n-re.

• **Szimulált hűtés algoritmusa:** A következő csúcs választása véletlenszerű. Ha a kiválasztott csúcs (r) célfüggvény-értéke jobb, mint az aktuális csúcsé (n), akkor odalép, ha rosszabb, akkor az új csúcs elfogadásának valószínűsége fordítottan arányos f(n) – f(r) különbséggel. Ez az arány ráadásul folyamatosan változik a keresés során: ugyanolyan

különbség esetén kezdetben nagyobb, később kisebb valószínűséggel fogja a rosszabb értékű r csúcsot választani.

28. Visszalépéses keresés

A startcsúcsból az aktuális csúcsba vezető utat (és az arról leágazó még ki nem próbált éleket) tartja nyilván (globális munkaterületen), a nyilvántartott út végéhez egy új (ki nem próbált) élt fűzhet vagy a legutolsó élt törölheti (visszalépés szabálya), a visszalépést a legvégső esetben alkalmazza. A visszalépés teszi lehetővé azt, hogy egy korábbi továbblépésről hozott döntés megváltozhasson. Ez tehát egy módosítható vezérlési stratégia. A keresésbe sorrendi és vágó heurisztika építhető. Mindkettő lokálisan, az aktuális csúcsból kivezető, még ki nem próbált élekre vonatkozik. Visszalépés feltételei: zsákutca, zsákutca torkolat, kör, mélységi korlát.

- VL1 (nincs kör- és mélységi korlát figyelés) véges körmentes irányított gráfokon terminál, és ha van megoldás, akkor talál egyet.
- VL2 (általános) δ-gráfokon terminál, és ha van megoldás a mélységi korláton belül, akkor talál egyet.

Könnyen implementálható, kicsi memória igényű, mindig terminál, és ha van (a mélységi korlát alatt), akkor megoldást talál. De nem garantál optimális megoldást, egy kezdetben hozott rossz döntést csak nagyon sok lépés után képes korrigálni és egy zsákutca-szakaszt többször is bejárhat, ha abba többféle úton is el lehet jutni.

29. Gráfkeresés

A globális munkaterületén a startcsúcsból kiinduló már feltárt utak találhatók (ez az ún. kereső gráf), külön megjelölve az utak azon csúcsait, amelyeknek még nem (vagy nem eléggé jól) ismerjük a rákövetkezőit. Ezek a nyílt csúcsok. A keresés szabályai egy nyílt csúcsot terjesztenek ki, azaz előállítják (vagy újra előállítják) a csúcs összes rákövetkezőjét. A vezérlési stratégia a legkedvezőbb nyílt csúcs kiválasztására törekszik, ehhez egy kiértékelő függvényt (f) használ. Mivel egy nyílt csúcs, amely egy adott pillanatban nem kerül kiválasztásra, később még kiválasztódhat, ezért itt egy módosítható vezérlési stratégia valósul meg.

A keresés minden csúcshoz nyilvántart egy odavezető utat (π visszamutató pointerek segítségével), valamint az út költségét (g). Ezeket az értékeket működés közben alakítja ki, amikor a csúcsot először felfedezi vagy később egy olcsóbb utat talál hozzá. Mindkét esetben (amikor módosultak a csúcs ezen értékei) a csúcs nyílttá válik. Amikor egy már korábban kiterjesztett csúcs újra nyílt lesz, akkor a már korábban felfedezett leszármazottainál a visszafelé mutató pointerekkel kijelölt út költsége nem feltétlenül egyezik majd meg a nyilvántartott g értékkel, és az sem biztos, hogy ezek az értékek az eddig talált legolcsóbb útra vonatkoznak.

- Nem-informált gráfkeresések: mélységi gráfkeresés (f = -g, minden (n,m) élre c(n,m)=1), szélességi gráfkeresés (f = g, c(n,m)=1), egyenletes gráfkeresés (f = g)
- Heurisztikus gráfkeresések f-je a h a heurisztikus függvényre épül, amely minden csúcsban a hátralevő optimális h* költséget becsli. Ilyen az előre tekintő gráfkeresés (f = h), az A algoritmus (f = g+h, h≥0), az A* algoritmus (f = g+h, h*≥ h≥0 – h megengedhető), az A C algoritmus (f = g+h, h*≥h≥0, minden (n,m) élre h(n)-h(m)≤c(n,m)), és B algoritmus (ahol az f= g+h, h≥0 helyett a g-t használjuk a kiterjesztendő csúcs kiválasztására azon nyílt csúcsok közül, amelyek f értéke kisebb, mint az eddig kiterjesztett csúcsok f értékeinek maximuma).

Véges δ-gráfokon minden gráfkeresés terminál, és ha van megoldás, talál egyet. A nevezetes gráfkeresések többsége végtelen nagy gráfokon is találnak megoldást, ha van megoldás. (Kivétel az előre-tekintő keresés és a mélységi korlátot nem használó mélységi gráfkeresés.)

Az A*, A ^{C} algoritmusok optimális megoldást találnak, ha van megoldás. Az A ^{C} algoritmus egy csúcsot legfeljebb egyszer terjeszt csak ki.
Egy gráfkeresés memória igényét a kiterjesztett csúcsok számával, futási idejét ezek kiterjesztéseinek számával mérjük. (Egy csúcs általában többször is kiterjesztődhet, de δ-gráfokban csak véges sokszor.) A*algoritmusnál a futási idő legrosszabb esetben exponenciálisan függ a kiterjesztett csúcsok számától, de ha olyan heurisztikát választunk, amelyre már A ^{C} algoritmust kapunk, akkor a futási idő lineáris lesz. Persze ezzel a másik heurisztikával változik a kiterjesztett csúcsok száma is, így nem biztos, hogy egy A C algoritmus ugyanazon a gráfon összességében kevesebb kiterjesztést végez, mint egy csúcsot többször is kiterjesztő A*algoritmus. A B algoritmus futási ideje négyzetes, és ha olyan heurisztikus függvényt használ, mint az A*algoritmus (azaz megengedhetőt), akkor ugyanúgy optimális megoldást talál (ha van megoldás) és a kiterjesztett csúcsok száma (mellesleg a halmaza is) megegyezik az A*algoritmus által kiterjesztett csúcsokéval.

30. Általános gráfkereső algoritmus

Jelölések:

– keresőgráf (G) : a reprezentációs gráf eddig bejárt és eltárolt része

– nyílt csúcsok halmaza (NYÍLT) : kiterjesztésre várakozó csúcsok, amelyeknek gyerekeit még nem vagy nem eléggé jól ismerjük

– kiterjesztett csúcsok halmaza (ZÁRT) : azok a csúcsok, amelyeknek a gyerekeit már előállítottuk

– kiértékelő függvény (f: NYÍLT → ℝ) : kiválasztja a megfelelő nyílt csúcsot kiterjesztésre

31. Gráfkeresés függvényei

π: N → N szülőre visszamutató pointer

- π(n) = n csúcs már ismert szülője, π(start) = nil
  • π egy start gyökerű irányított feszítőfát jelöl ki G-ben: π-feszítőfa, π-út
  Jó lenne ha a π-út optimális start→n G-beli utat jelölne ki: a π feszítőfa optimális lenne.

- g: N → ℝ költség függvény – g(n) = cα(start,n) – egy már megtalált α ∈{start→n} út költsége •
  Jó lenne ha minden n-re a g(n) a π-út költségét mutatná, azaz π és g konzisztens lenne. •

Az *n* csúcs akkor ***korrekt***, ha konzisztens és optimális:
$g(n) = c^{\pi}(start,n)$ és $c^{\pi}(start,n) = \min_{\alpha \in \{start \to n\} \cap G} c^{\alpha}(start,n)$
A *G* korrekt, ha minden csúcsa korrekt.

32. Csökkenő kiértékelő függvény

- Egy GK kiértékelő függvénye csökkenő, amennyiben a egy csúcs kiértékelő függvény értéke az algoritmus működése során nem növekszik, viszont mindig csökken, valahányszor a korábbinál olcsóbb utat találunk hozzá.
- Csökkenő kiértékelő függvény mellett a GK időről időre automatikusan helyreállítja a kereső gráf korrektségét, azaz a π feszítő fájának optimálisságát és konzisztenciáját.

33. Nevezetes gráfkereső algoritmusok

Nem-informált:

-mélységi (MGK)
-szélességi (SZGK)
-egyenletes (EGK)

Heurisztikus

-előre tekintő (mohó, best-first)
-A, A* , Ac
-A**, B

34. Nevezetes nem-informált algoritmusok

Algoritmus	Definíció	Eredmények
Mélységi gráfkeresés (MGK)	$f = -g$, $c(n,m) = 1$	végtelen gráfokban csak mélységi korláttal garantál megoldást
Szélességi gráfkeresés (SZGK)	$f = g$, $c(n,m) = 1$	optimális (legrövidebb) megoldást adja, ha van (még végtelen δ-gráfokban is) egy csúcsot legfeljebb egyszer terjeszt ki
Egyenletes gráfkeresés (EGK)	$f = g$	optimális (legolcsóbb) megoldást adja, ha van (még végtelen δ-gráfokban is) egy csúcsot legfeljebb egyszer terjeszt ki

35. Heurisztika a gráfkereséseknél

Heurisztikus függvénynek nevezzük azt a h:N→ℝ függvényt, amelyik egy csúcsnál megbecsüli a csúcsból a célba vezető („hátralévő") optimális út költségét.

$h(n) \approx \min_{t \in T} c^*(n,t) = c^*(n,T) = h^*(n)$

36. Heurisztikus függvények tulajdonságai

Nevezetes tulajdonságok:

-Nem-negatív: $h(n) \geqq 0 \ \forall n \in N$

-Megengedhető (admissible): $h(n) \leqq h^*(n) \ \forall n \in N$

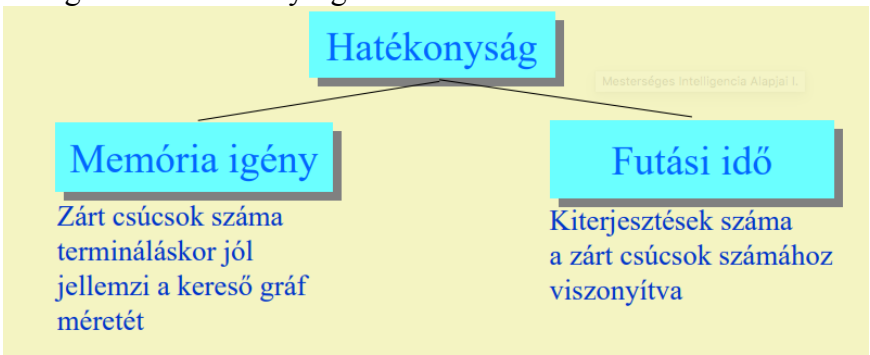
-Monoton megszorítás: $h(n) - h(m) \leqq c(n,m) \ \forall (n,m) \in A$ (következetes)

37. Nevezetes heurisztikus algoritmusok

Algoritmus	Definíció	Eredmények
Előre tekintő gráfkeresés	$f = h$	nincs említhető extra tulajdonsága
A algoritmus	$f = g+h$ és $h \geqq 0$	• megoldást ad, ha van megoldás (még végtelen δ-gráfban is)
A algoritmus*	$f = g+h$ és $h \geqq 0$ és $h \leqq h^*$	• optimális megoldást ad, ha van (még végtelen δ-gráfban is)
A^c algoritmus	$f = g+h$ és $h \geqq 0$ és $h \leqq h^*$ és $h(n) - h(m) \leqq c(n,m)$	• optimális megoldást ad, ha van (még végtelen δ-gráfban is) • egy csúcsot legfeljebb egyszer terjeszt ki

egyenletes gráfkeresés: $f = g+0$

37. A* algoritmus hatékonysága



38. B algoritmus

A B algoritmust az A algoritmusból kapjuk úgy, hogy bevezetjük az F aktuális küszöbértéket, majd

– az 1. lépést kiegészítjük az F := f(s) értékadással,

– a 4. lépést pedig helyettesítjük az
if minf (NYÍLT) <F
then n := min $_g$ (m∈NYÍLT | f(m) ‹ F)
else n := min $_f$ (NYÍLT); F := f(n)
endif elágazással.

39. B algoritmus futási ideje

A B algoritmus ugyanúgy működik, mint az A* , azzal a kivétellel, hogy egy árokhoz tartozó csúcsot csak egyszer terjeszt ki.

Futási idő elemzése:

• Legrosszabb esetben
– minden zárt csúcs először küszöbcsúcsként terjesztődik ki. (Csökkenő kiértékelő függvény mellett egy csúcs csak egyszer, a legelső kiterjesztésekor lehet küszöb.)
– Az i-dik árok legfeljebb az összes addigi i–1 darab küszöbcsúcsot tartalmazhatja (a start csúcs nélkül).
• Így az összes kiterjesztések száma legfeljebb ½·k ^{2}

40. Milyen a jó heurisztika?

-megengedhető: h(n) ≤ h * (n)
-jól informált: h(n) ~ h* (n)
– monoton megszorítás: h(n)– h (m) ≤ c(n,m)

Változó heurisztikák:

– f=g+ϕ·h ahol ϕ~d
– B' algoritmus

41. B' algoritmus

if $h(n)<min_{m\in\Gamma(n)}\,(c(n,m)+h(m))$

then $h(n):=min_{m\in\Gamma(n)}\,(c(n,m)+h(m))$

else **for** $\forall m\in\Gamma(n)$-re **loop**

if $h(n)- h(m)>c(n,m)$ **then** $h(m) := h(n)- c(n,m)$

endloop

• A h megengedhető marad
• A h nem csökken
• A mononton megszorításos élek száma nő

42. Mohó A algoritmus

Nincs mindig szükség az optimális megoldásra.
– Ilyenkor a mohó A* algoritmus is használható, amely rögtön megáll, ha célcsúcs jelenik meg a NYÍLT-ban.

A mohó A* algoritmus csak a megoldás megtalálását garantálja. De belátható
– Ha h megengedhető és ∀t∈T: ∀(n,t)∈A: h(n)+α≥c(n,t), akkor a talált megoldás költsége: g(t)≤h * (s)+α

A mohó A* algoritmus megengedhető heurisztika mellett akkor garantálja az optimális megoldást is,
– ha∀t∈T: ∀(n,t)∈A: h(n)=c(n,t) vagy
– ha h monoton és ∃α≥0: ∀t∈T: ∀(n,t)∈A: h(n)+α=c(n,t)

43. Minimax algoritmus

1. A játékfának az adott állás csúcsából leágazó részfáját felépítjük néhány szintig.
2. A részfa leveleit kiértékeljük aszerint, hogy azok számunkra kedvező, vagy kedvezőtlen állások.

3. Az értékeket felfuttatjuk a fában:
• A saját (MAX) szintek csúcsaihoz azok gyermekeinek maximumát: szülő := max (gyerek$_1$,..., gyerek $_k$)
• Az ellenfél (MIN) csúcsaihoz azok gyermekeinek minimumát: szülő := min (gyerek$_1$,..., gyerek $_k$)
4. Soron következő lépésünk ahhoz az álláshoz vezet, ahonnan a gyökérhez felkerült a legnagyobb érték.

44. Átlagoló kiértékelés
Célja a kiértékelő függvény esetleges tévedéseinek simítása.

45. Váltakozó mélységű kiértékelés
Célja, hogy a kiértékelő függvény minden ágon reális értéket mutasson. Nem megbízható ez az érték abban a csúcsban, amelynek szülőjénél a kiértékelő függvény lényegesen eltérő értéket mutat. Egy adott szintig (minimális mélység) mindenképpen felépítjük a részfát, majd ettől a szinttől kezdve egy adott szintig (maximális mélység) csak azon csúcsokat terjesztjük ki, amelyekre nem teljesül a nyugalmi teszt: $|f(\text{szülő}) - f(\text{csúcs})| < K$,

46. Szelektív kiértékelés
Célja a memória-igény csökkentése. Elkülönítjük a lényeges és lényegtelen lépéseket, és csak a lényeges lépéseknek megfelelő részfát építjük fel. Ez a szétválasztás heurisztikus ismeretekre épül.

47. Negamax algoritmus
Negamax eljárást könnyebb implementálni. – Kezdetben (–1)-gyel szorozzuk azon levélcsúcsok értékeit, amelyek az ellenfél (MIN) szintjein vannak, majd – Az értékek felfuttatásánál minden szinten az alábbi módon számoljuk a belső csúcsok értékeit:
szülő := max([?] gyerek1 ,..., [?] gyerek $_k$)

48. Alfa-béta algoritmus
Visszalépéses algoritmus segítségével járjuk be a részfát (olyan mélységi bejárás, amely mindig csak egy utat tárol). Az aktuális úton fekvő csúcsok ideiglenes értékei:
– a MAX szintjein α érték: ennél rosszabb értékű állásba innen már nem juthatunk
– A MIN szintjein β érték: ennél jobb értékű állásba onnan már nem juthatunk
Lefelé haladva a fában $\alpha := -\infty$, és $\beta := +\infty$.
Visszalépéskor az éppen elhagyott (gyermek) csúcs értéke (felhozott érték) módosíthatja a szülő csúcs értékét:
– a MAX szintjein: $\alpha :=$ max(felhozott érték, α)
– a MIN szintjein: $\beta :=$ min(felhozott érték, β)
Vágás: ha az úton van olyan α és β, hogy $\alpha \geq \beta$.

49. Kétszemélyes (teljes információjú, zéró összegű, véges) játékok
A játékokat állapottér-reprezentációval szokás leírni, és az állapot-gráfot faként ábrázolják. A győztes (vagy nem-vesztes) stratégia egy olyan elv, amelyet betartva egy játékos az ellenfél minden lépésére tud olyan választ adni, hogy megnyerje (ne veszítse el) a játékot. Valamelyik játékosnak biztosan van győztes (nem-vesztes) stratégiája. Győztes (nem-vesztes) stratégia keresése a játékfában kombinatorikus robbanást okozhat, ezért e helyett részfa kiértékelést szoktak alkalmazni a soron következő jó lépés meghatározásához. A minimax algoritmus az aktuális állásból felépíti a játékfa egy részét, kiértékeli annak leveleit aszerint, hogy azok által képviselt állások milyen mértékben kedveznek nekünk vagy az ellenfélnek, majd szintenként váltakozva az ellenfél szintjein a gyerekcsúcsok értékeinek minimumát, a saját szintjeinken azok maximumát futtatjuk fel a szülőcsúcshoz. Ahonnan a gyökérhez kerül érték, az lesz soron következő lépésünk. A minimax legismertebb módosítása az alfa-béta algoritmus, amely

egyfelől kisebb memória igényű (egyszerre csak egy ágat tárol a vizsgált részfából), másfelől egy sajátos vágási stratégia miatt jóval kevesebb csúcsot vizsgál meg, mint a minimax.

50. Populáció

A problémára adható néhány lehetséges választ, azaz a problématér több egyedét tároljuk egyszerre. Ez a populáció. θ Kezdetben egy többnyire véletlen populációt választunk. A cél egy bizonyos célegyed vagy egy jó populáció előállítása. θ Az egyedeket egy rátermettségi függvény alapján hasonlítjuk össze. θ A populációt lépésről lépésre javítjuk úgy, hogy a kevésbé rátermett egyedek egy részét a rátermettebbekhez hasonló egyedekre cseréljük le. Ez a változtatás visszavonhatatlan. Ez tehát egy nem-módosítható stratégiájú keresés.

51. Evolúciós operátorok és a terminálási feltétel

Szelekció: Kiválasztunk néhány (lehetőleg rátermett) egyedet szülőnek.
Rekombináció (keresztezés): Szülőkből utódok készülnek úgy, hogy a szülők tulajdonságait öröklöjék az utódok.
Mutáció: Az utódok tulajdonságait kismértékben módosítjuk.
Visszahelyezés: Új populációt alakítunk ki az utódokból és a régi populációból.
Terminálási feltétel:
• ha a célegyed megjelenik a populációban
• ha a populáció egyesített rátermettségi függvény értéke egy ideje nem változik.

52. Evolúció alapalgoritmusa

```
Procedure EA
populáció := kezdeti populáció
while terminálási feltétel nem igaz loop
        szülők := szelekció(populáció)
        utódok := rekombináció( szülők )
        utódok := mutáció(utódok)
        populáció := visszahelyezés(populáció, utódok)
endloop
```

53. Evolúciós algoritmus elemei

problématér egyedeinek reprezentációja: **kódolás**
rátermettségi függvény (fitnesz függvény)
– kapcsolat a kódolással és a céllal
evolúciós operátorok
– szelekció, rekombináció, mutáció, visszahelyezés
kezdő populáció, megállási feltétel (cél)
stratégiai paraméterek
– populáció mérete, mutáció valószínűsége, utódképzési ráta, visszahelyezési ráta, stb.

54. Szelekció

Célja: a rátermett egyedek kiválasztása úgy, hogy a rosszabbak kiválasztása is kapjon esélyt. **– Rátermettség arányos** (rulett kerék algoritmus): minél jobb a rátermettségi függvényértéke egy elemnek, annál nagyobb valószínűséggel választja ki
– Rangsorolásos: rátermettség alapján sorba rendezett egyedek közül a kisebb sorszámúakat nagyobb valószínűséggel választja ki
– Versengő: véletlenül kiválasztott egyedcsoportok (pl. párok) legjobb egyedét választja ki.
– Csonkolásos v. selejtezős: a rátermettség szerint legjobb (adott küszöbérték feletti) valahány egyedből véletlenszerűen választ néhányat.

55. Rekombináció

A feladata az, hogy adott szülő-egyedekből olyan utódokat hozzon létre, amelyek a szüleik tulajdonságait "öröklik".

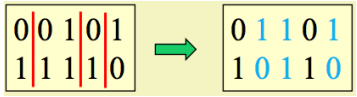
– Keresztezés: véletlen kiválasztott pozíción jelcsoportok (gének) vagy jelek cseréje

– Rekombináció: a szülő egyedek megfelelő jeleinek kombinálásával kapjuk az utód megfelelő jelét.

56. Keresztezés

Egy- illetve többpontos keresztezés

– Kódszakaszokat cserélünk



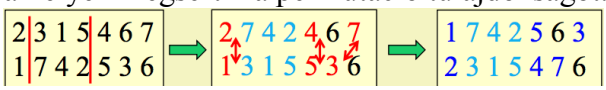
Egyenletes keresztezés

– Jeleket cserélünk

57. Permutációk keresztezése

Parciálisan illesztett keresztezés

– Egy szakasz cseréje után párba állítja és kicseréli azokat a szakaszon kívüli elemeket, amelyek megsértik a permutáció tulajdonságot.



Ciklikus keresztezés 1.

1.Választ egy véletlen i ∈ [1..length] -t

2. ai ↔ bi

3. Keres olyan j ∈ [1..length] -t (j≠i), amelyre $a_j = a_i$,

4. Ha nem talál, akkor vége, különben i:=j

5. goto 2.

57. Rekombináció vektorokra

Köztes rekombináció

– A szülők (x, y) által kifeszített hipertégla környezetében lesz az utód (u).

– $\forall i=1\ldots n : u_i = a_i x_i + (1-a_i) y_i$ $a_i \in [-h, 1+h]$ véletlen

Lineáris rekombináció

– A szülők (x, y) által kifeszített egyenesen a szülők környezetében vagy a szülők között lesz az utód (u).

– $\forall i=1\ldots n : u_i = ax_i + (1-a)y_i$ $a \in [-h, 1+h]$ véletlen

58. Mutáció

A mutáció egy egyed (utód) kis mértékű véletlen változtatását végzi.

Valós tömbbel való kódolásnál kis p valószínűséggel:

– $\forall i=1\ldots n : z_i = x_i \pm range_i * (1-2*p)$

Bináris tömbbel való kódolásnál kis p valószínűséggel:

– $\forall i=1\ldots n : z_i = 1 - x_i$ if random[0..1]

Permutáció esetén

– egy jelpár cseréje

– egy kódszakaszon a jelek ciklikus léptetése vagy megfordítása vagy átrendezése.

59. Visszahelyezés

A visszahelyezés a populációnak az utódokkal történő frissítése: Kiválasztja a populációnak a lecserélendő egyedeit, és azok helyére a kiválasztott utódokat teszi

utódképzési ráta (u) =utódok száma / populáció száma

visszahelyezési ráta (v) =lecserélendő egyedek száma / populáció száma

-ha u=v, akkor feltétlen cseréről van szó

- ha u<v, akkor utód több példánya is bekerülhet
 – ha uv, akkor az utódok közül szelektál