

MI vizsgák

Alant olvasható néhány GT-féle Mesterséges Intelligencia vizsga, a kérdések alatt az én megoldásaimmal.

FONTOS!

Gyakorlati feladatokat nem tartalmaz!

A tartalom helyességére semmilyen garanciát nem vállalok!

Az algoritmusokat leíró pszeudokód részeket java szintaxissal írom le.

A rajzoló részeket igyekeztem ASCII-ban szépen megrajzolni több, de inkább kevesebb sikerrel.

Ahol egy példát, sort, stb. kér egy valamire, ott általában leírom az összeset, hogy meg lehessen tanulni az összeset (ld. táblázatok).

Vizsgák

2012.06.20(?)

2012.06.13

2012.06.06

2012.05.30

2012.05.23

2015.05.28

2015.06.03

2016.06.22

2012.06.20(?)

Elméleti rész

- **Milyen hatással van a heurisztika általában a kereső rendszerek működésére?**
A heurisztika olyan, a feladathoz kapcsolódó ötlet, amit közvetlenül építünk be egy algoritmusba azért, hogy annak eredményessége és hatékonysága javuljon, bár erre semmiféle garanciát nem ad.
- **Írja le a hegymászó algoritmust!**
Olyan lokális keresési algoritmus, ami a *start* csúcsból kiindulva mindig az aktuális csúcs legjobb gyerekére lép, ami lehetőleg nem a szülője. Algoritmusa:

```
public Node hegymaszo(Node start){
    Node akt = start;

    while (!T.contains(akt)){
        if (akt.children() == null) return null; // nincs megoldás
        if (akt.children() - akt.parent() == null){
            akt = akt.parent();
        }
    }
}
```

```

    } else {
        akt = getBestChild(akt.children() - akt.parent());
    }
}

return akt;
}

```

- **Mit tesz az általános gráfkereső algoritmus akkor, amikor egy már korábban felfedezett csúcshoz talál minden addiginál olcsóbb utat?**
Ha $m \in G$ és $g(m) < g(n) + c(n, m)$, akkor:
 $\pi(m) = n$ és $g(m) = g(n) + c(n, m)$, és az m csúcsot nyílttá tesszük. Ha a kiértékelő függvény csökkenő, a korrektség megától helyreáll.
- **Mit tartalmaz egy probléma dekompozíciós reprezentációja?**
 - A feladat részproblémáinak általános leírása
 - A kiinduló probléma
 - Az egyszerű problémák, amelyikről könnyen eldönthető, hogy megoldhatóak-e, vagy sem
 - A dekomponáló műveleteket:
 $D : \text{probléma} \rightarrow \text{probléma}^+$
 $D(p) = \langle p_1, \dots, p_n \rangle$
- **Adjon példát legalább három rekombinációs operátorra!**
 - Egy pontos keresztezés
 - Több pontos keresztezés
 - Egyenletes keresztezés
 - Parciálisan illesztett keresztezés
 - Ciklikus keresztezés

Gyakorlati rész

1. **Fekete-fehér kirakó állapotér-reprezentációja. Problématér és állapotér méretének becslése.**
2. **Szemléltesse az A algoritmus működését! (Megvolt adva egy gráf, egy hisztogram, egy táblázat. A hisztogramot és a táblázatot kellett megfelelően kitölteni)**

2012. 06. 13.

Elméleti rész

- **Melyik problémáját küszöböli ki a tabu keresés a hegymászó módszernek + probléma jellemzése + tabu megoldásának jellemzése**
A hegymászó algoritmus lokális optimum hely körül, vagy ekvidisztans felületen lévő körön végtelen működésbe eshet. Ezt küszöböli ki a tabu keresés.
Nyilvántartjuk az optimális csúcsot, és a tabu halmazt. Minden lépésben:
 - Az aktuális csúcs legjobb gyereke lép, ami nincs a tabu halmazban
 - Ha az aktuális csúcs jobb, mint az optimális, akkor $opt = akt$;
 - akt -ot hozzáadja a tabu halmazhoz.

Algoritmus:

```

public Node tabu(Node start){
    Node akt = start;
    Node opt = start;
    Node[] tabu = new Node[5];

    while(!T.contains(akt) || opt.hasNotChangedForALongTime()){
        akt = getBestChild(akt.children() - tabu); // akt legjobb gyereke
        updateTabu(akt);                          // akt is tabu lesz
        if (akt.value() > opt.value()){            // akt jobb, mint opt
            opt = akt;
        }
    }
    return akt;
}

```

- **Visszalépéses keresés munkaterülete, keresési szabályai, vezérlési stratégia**
 - Munkaterülete: egy út az aktuális csúcsba a startcsúcsból + a leágazó, még ki nem próbált élek
 - Keresés szabályai: a nyilvántartott úthoz egy új él hozzáfűzése, vagy az utolsó él törlése (visszalépés)
 - Vezérlési stratégia: A visszalépés szabályát csak legvégső esetben alkalmazza
 - A visszalépés feltételei:
 - Zsákutca
 - Zsákutca torkolat
 - Kör
 - Mélységi korlát
- **Általános gráf ker. eredményei**
 1. δ -gráfokban egy csúcsot véges sokszor terjeszt ki
 2. véges δ -gráfban terminál
 3. véges δ -gráfban, ha van megoldás, megtalálja és terminál
- **Mi a hiperút?**
 $n^a \rightarrow M$ hiper-út, ($n \in N, M \in N^+$) olyan véges részgráf, amiben:
 - M csúcsaiból nem indul hiper-él
 - M -en kívüli csúcsból csak 1 db hiper-él indul
 - Minden csúcs elérhető az n csúcsból egy közösirányított úton
 - Hossza az éleinek a hossza, költsége definiálható
- **Mikor jó egy szelekció? + példa**
Célja a rátermett egyedek kiválasztása úgy, hogy rosszabbak kiválasztása is kapjon esélyt

Gyakorlati rész

1. Állapottér rep. SAT problémára
2. Minimax

2012. 06. 06.

Elméleti rész

- **Milyen eredményre képes a visszalépéses keresés első, illetve második változata?**
 - *VLI*: Zsákutca és zsákutca torkolat visszalépési szabályokat implementáljuk
 - Véges körmentes gráfon mindig terminál és talál megoldást, ha van

- VL2: A fenti kettőn kívül implementálja a kör és a mélységi korlát visszalépési feltételeket is
 - Véges δ -gráfon mindig terminál, és talál megoldást, ha van olyan, ami a mélységi korlátnál rövidebb
- Mikor nevezünk egy gráfkereső algoritmust szélességi keresésnek? Milyen állítást mondhatunk ki vele kapcsolatban?**

Elnevezés	Definíció	Eredmények
Mélységi	$f=g,$ $c(n,m)=1$	Végtelen gráfokban mélységi korláttal megoldást garantál
Szélességi	$f=g,$ $c(n,m)=1$	Végtelen gráfokban a legrövidebb megoldást adja, egy csúcsot csak egyszer terjeszt ki.
Egyenletes	$f=g$	Végtelen gráfokban a legolcsóbb megoldást adja, egy csúcsot legfeljebb egyszer terjeszt ki.

- Sorolja fel, milyen módosításait ismerte meg a minimax algoritmusnak, és írja melléjük, hogy ezek milyen szempontból javítanak annak működésén?**
 - Átlagoló kiértékelés
 - A kiértékelő függvény esetleges tévedéseit simítja ki
 - Váltakozó mélységű kiértékelés
 - A kiértékelő függvény minden ágon reális értéket mutat
 - Szelektív kiértékelés
 - A memóriaigényt csökkenti (csak a lényeges lépéseket értékeli)
- Hogyan történik az evolúciós algoritmusokban a visszahelyezés?**
 A populáció az utódokkal történő frissítése: kiválasztja a populáció lecserélendő egyedeit, és azok helyére az utódokat teszi.
 $utódképzési\ ráta(u) = utódok\ száma / populáció\ száma$
 $visszahelyezési\ ráta(v) = lecserélendő\ egyedek\ száma / populáció\ száma$
 - Ha $u=v$: feltétlenül csere
 - Ha $u < v$: egy utód több példányban is bekerülhet
 - Ha $u > v$: az utódok közül szelektál
- A rezolúció módszere egy speciális kereső rendszer. Írja le, hogy ebben mik lesznek a keresőrendszer fő részei!**
 - Globális munkaterület -> aktuális klózhalmaz
 - Kiindulási érték -> "axiómák -> célállítás" klózzai
 - Terminálási feltétel -> "sikeres" – üres klóz; "sikertelen" – nincs újabb rezolvens klóz
 - Kereső szabály -> rezolvens képzés
 - Vezérlési stratégia -> nem módosítható
 - Heurisztika -> nincs

Gyakorlati rész

- Állapottér repr. a misszionárius – kannibál problémára
- (2,2) átlagoló eljárás egy adott játékfára. Soron következő lépés.

2012. 05. 30.

Elméleti rész

- **Osztályozza a vezérlési stratégiákat!**

Háromféle vezérlési stratégiát különböztetünk meg:

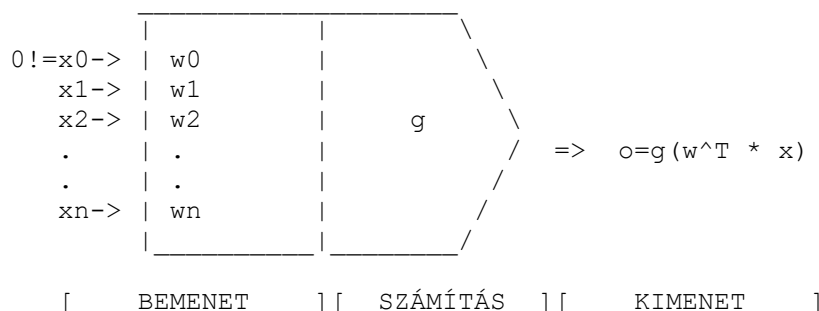
- Elsődleges vezérlési stratégiák: Független a feladattól, nem merít a feladat ismereteiből, sem a modell sajátosságaiból. Fajtái:
 - Nem módosítható, pl. lokális keresések, evolúciós algoritmus, rezolúció
 - Módosítható, pl. visszalépéses keresések, gráfkeresés
- Másodlagos vezérlési stratégiák: Nem függ a feladattól, de épít a modell sajátosságainak ismeretére
- Heurisztikák: A feladattól származó, annak modelljében nem rögzített, a megoldást segítő speciális ismeret
- **Melyek a visszalépéses keresés előnyei hátrányai**
 - Előnyök:
 - Mindig terminál, talál megoldást, de csak a mélységi korláton belül
 - Könnyen implementálható
 - Kicsi a memóriaigénye
 - Hátrányok:
 - Nem ad optimális megoldást (iterációba szervezhető)
 - Kezdetben hozott rossz döntést csak sok visszalépés korrigál
 - Egy zsákutca rész többször is bejárhat a keresés
- **Mikor nevezünk egy gráf kereső algoritmust A^c algoritmusnak, és melyek a legfőbb tulajdonságai**

Elnevezés	Definíció	Eredmények
Előre tekintő gráfkeresés	$f=h$	Nincs említésre méltó tulajdonsága
A algoritmus	$f=g+h, h \leq h^*$	Megoldást ad, ha van, még végtelen gráfban is
A^* algoritmus	$f=g+h, h \leq h^*, h \geq 0$	Optimális megoldást ad, ha van, még végtelen gráfokban is
A^c algoritmus	$f=g+h, h \leq h^*, h \geq 0, h(n)-h(m) \leq c(n,m)$	Optimális megoldást ad, ha van és ugyanazt a csúcsot nem terjeszti ki kétszer

- **És/Vagy gráf**

Olyan gráf ($R=(N,A)$), ahol N -beli csúcsok részproblémákat jelölnek s startcsúcsból kiindulva, megoldható egyszerű problémák a célcsúcsok (T). A egy élköteg (hiper-él), ami egy dekomponáló művelet hatását mutatja, a dekomponálódó probléma csúcsából a dekomponálással előállított részproblémák csúcsaiba vezet.

- **Hogyan néz ki egy általánosított perceptron, és hogyan számolja ki a kimeneti értéket**



Az $x_0 \neq 0$ mert az a stimuláló inger: aktivizációs függvény.

A perceptron összegzett bemenetéről tanult képlet:

$$I = \sum_{i=0}^n w_i x_i = \mathbf{w}^T \mathbf{x}$$

Gyakorlati rész

1. **Állapottár reprezentáció n királynő**
2. **Visszalépéses keresés lejátssza egy gráfon, mennyi visszalépés volt, mi a megoldási út, számozni kellett bejárás szerinti sorrendbe, és x-el jelölni azokat a csúcsokat ahol volt visszalépés**

2012. 05. 23.

Elméleti rész

- **Nevezze meg és jellemezze a keresőrendszerek fő részeit.**
 - *Globális munkaterület:* tárolja a keresés során megszerzett és megőrzött ismeret (azaz egy részgráfot)
 - *Vezérlési stratégia:* Végrehajtható szabályok alapján kiválaszt egy megfelelő (általános elv + heurisztika)
 - *Keresési szabályok:* Megváltoztatják a globális munkaterület tartalmát (előfeltétel és hatás)

Keresőrendszer általános algoritmusa:

```
public Data KR (Data start){
    Data data = start;

    while(!terminate(data)){
        Rule r = rules.getApplicableRule(data);
        data = r.apply(data);
    }

    return data;
}
```

- **Melyek a visszalépés feltételei a visszalépéses algoritmus legáltalánosabb változatában?**
VL1-ben:
 - Zsákutca szabály: egyértelmű :)
 - Zsákutca torkolat szabály: olyan csúcsból lép vissza, aminek minden gyereke zsákutca
- **Mikor Nevezünk egy gráfkereső algoritmust A* algoritmusnak és mit tudunk ennek az eredményességéről.**

Elnevezés	Definíció	Eredmények
Előre tekintő gráfkeresés	$f=h$	Nincs említésre méltó tulajdonsága

Elnevezés	Definíció	Eredmények
A algoritmus	$f=g+h, h \leq h^*$	Megoldást ad, ha van, még végtelen gráfban is
A* algoritmus	$f=g+h, h \leq h^*, h \geq 0$	Optimális megoldást ad, ha van, még végtelen gráfokban is
A ^c algoritmus	$f=g+h, h \leq h^*, h \geq 0, h(n)-h(m) \leq c(n,m)$	Optimális megoldást ad, ha van és ugyanazt a csúcsot nem terjeszti ki kétszer

- **Mit jelent kétszemélyes játékoknál a nyerő stratégia és milyen állítást mondtunk ki ezzel kapcsolatban.**
 - Egy játékos nyerő stratégiája olyan elv, amit betartva az ellenfél minden lépésére tud olyan választ adni, hogy megnyerje a játékot
 - *Nem-vesztő stratégia:* A nyerő stratégia párja abban az esetben, ha a játék megengedi a döntetlent
 - A *nyerő/nem-vesztő* stratégia egy megfelelő ÉS/VAGY fa részgráffjával ábrázolható
 - Olyan részfa, ami egy olyan hiperút, amit a kezdő állás csúcsából vezet a nyerő/nem-vesztő végállások csúcsaiba
 - Tehát: keresése egy ÉS/VAGY gráfbeli *hiperút keresési probléma*.

TÉTEL: Ha a döntetlen nem megengedett, akkor a kétszemélyes kétszemélyes játékokban az egyik játékos számára biztosan létezik nyerő stratégia.
A kétszemélyes háromesélyes (~ahol megengedett a döntetlen) játékokban a nem vesztes stratégia lehet biztosan garantálni.

- **Mi történik egy evolúciós algoritmus egy iterációja során (milyen lépések hajtnak végre)**
 - *Szelekció:* Kiválasztunk néhány (lehetőleg rátermett) egyedet szülőnek:
`selection(Entity[] from)`
 - *Rekombináció:* A szülőkből utódok készülnek úgy, hogy a szülők tulajdonságait örököljék a gyerekek:
`recombination(Entity[] entityGroup)`
 - *Mutáció:* Az utódok tulajdonságait kismértékben módosítjuk
`mutation(Entity[] entityGroup)`
 - *Visszahelyezés:* Új populációalakítunk ki az utódokból és a régi populációból
`insertion(Entity[] to, Entity[] from)`

Az evolúció algoritmusa:

```
public Entity[] evolution(Entity[] initialPopulation) {
    Entity[] pop = initialPopulation;

    while (!terminate(pop)) {
        Entity[] parents = selection(pop);
        Entity[] offspring = recombination(parents);
        offspring = mutation(offspring);
        pop = insertion(pop, offspring);
    }

    return pop;
}
```

Gyakorlati rész

1. **Állapottér reprezentáció** utazó ügynökre (n darab város) és becsülje meg az állapottérnek és problémátérnek a méretét.
2. **Hegymászó algoritmus** Hanoi torony rep. gráfjában(ami a diákon is van) adott heurisztika mellett $\text{sum}(i=0\text{-tol } 3\text{-ig}) i \cdot v(i)$, ha két csucs között nem dont heurisztika akkor a baloldali csúcsot részesítsük előnyben.

2015.05.28

Elméleti rész

- **Elsődleges vezérlési stratégia osztályozása, jellemzése**
 - *Módosítható stratégiák:* Egy korábban meghozott rossz döntés megváltoztatható
pl. visszalépéses keresések, gráfkeresések
 - *Nem módosítható stratégiák:* Egy korábban meghozott rossz döntést a keresés során nem lehet megváltoztatni
pl. lokális keresés, evolúciós algoritmus, rezolúció
- **Probléma dekompozíció gráfrepresentációját összehasonlítani a visszafelé haladó kereséssel**
Fogalmam sincs...
- **Evolúciós algoritmus lépései és jellemzése, inicializálás, terminálás**
Az algoritmus lépései:
 - *Szelekció:* a populációból kiválasztunk néhány -- lehetőleg rátermett -- egyedet szülőknek
 - *Rekombináció:* a szülőkből gyerekek készülnek úgy, hogy mindkét szülő tulajdonságait megöröklik
 - *Mutáció:* az utódok tulajdonságait kis mértékben megváltoztatjuk
 - *Visszahelyezés:* új populációt alakítunk ki az utódokból és a régi populációból

Inicializálás: Kialakítjuk a kezdeti populációt

Terminálás: Addig futtatjuk az algoritmust, amíg el nem érjük a kívánt állapotot

Algoritmus:

```
public Entity[] evolution(Entity[] initialPopulation){
    Entity[] pop = initialPopulation;

    while(!terminate(pop)){
        Entity[] parents = selection(pop);
        Entity[] offspring = recombination(parents);
        offspring = mutation(offspring);
        pop = insertion(pop, offspring);
    }

    return pop;
}
```

- **Nyerő stratégia és az állítások hozzá**
Olyan elv, amit betartva a játékos az ellenfél minden lépésére tud olyan választ adni, hogy megnyerje a játékot. Ez nem egy konkrét győztes játszma, hanem olyan győztes játszmák összessége, amelyek közül egyet biztosan végig lehet játszani annak, aki rendelkezik a nyerő stratégiával.

- Döntetlent megengedő (azaz három esélyes) játékok esetén nem veszteső stratégiáról beszélünk
- Általános zéró összegű játékoknál: adott hasznosságot biztosító stratégia

TÉTEL: Ha a döntetlen nem megengedett, akkor a kétesélyes kétszemélyes játékokban az egyik játékos számára biztosan létezik nyerő stratégia. Három esélyes játékokban a nem veszteső stratégiát lehet garantálni.

- **Mélységi bejárás összehasonlítása több szempont alapján a VL2-vel szemben (stratégia, bejárás sorrendje, futási idő, stb...)**
 - *VL2:* élsorozatot tárol, az aktuális élhez mindig egy újabb élet fűz. Visszalép, ha zsákutcát, vagy kört talál vagy ha elérte a mélységi korlátját. Visszalépést csak a legvégső esetben alkalmaz.
 - *MB:* Gráfkeresés, tehát részgráfot tárol, egy utat a startcsúcsból. Egy útvégi csúcsot terjeszt ki, a kiértékelő függvénye alapján az éppen a legjobbnak véltet. Mélységi bejárást végez, mélységi korlát nélkül nem biztos, hogy talál megoldást.
 - *Mindkettő módosítható*, azaz egy kezdetben hozott rossz döntés kijavítható
 - *Futási idő:* Ha jó úton indulnak el hasonló, ha nem, akkor a gráfkeresés jobb, mert előbb érzékeli a "tévutakat"
- **Felügyelt és felügyelet nélküli tanulás def + példák**
Egy neuron esetében a Δw kiszámítása az éppen vizsgált minta bemeneti értékeire és a neuron által kiszámított kimenetre támaszkodik.
 - *Felügyelt tanulás* esetén felhasználjuk a neuronnal a vizsgált minta alapján elvárt kimeneti értékét is
 - *Felügyelet nélküli tanulás* esetén a várt kimenetre nincs szükség

Példák:

Felügyelt tanulás (Delta szabály) Felügyelet nélküli tanulás (Hebb szabály)

$$\Delta w_{ij} = \eta \cdot o_i \cdot (t_j - o_j) \qquad \Delta w_{ij} = \eta \cdot o_i \cdot o_j$$

Jelmagyarázat:

η : tanulási együttható, a tanulás sebességét befolyásolja.

o_i : az i -edik neuron számított kimenete, és egyben a j -edik neuron i -edik bemenete is.

w_{ij} : a j -edik neuron bemenetének súlya

o_j : a j -edik neuron számított kimenete

t_j : a j -edik neuron várt kimenete

Gyakorlati rész

3. **Állapottér reprezentáció: Tic Tac Toe**
4. **2-2 átlagoló kiértékelés**
5. **A algoritmus egy megadott gráfon**

2015.06.03.

Elméleti rész

- **Heurisztika, mire jó, KR-ben hol és mire használjuk?**

Feladattól származó, annak modelljében nem rögzített, a megoldást segítő speciális ismeret. Közvetlenül építjük be az algoritmusba, hogy annak hatékonysága és eredményessége javuljon, habár erre semmiféle garanciát nem nyújt. A hatékonyság növelése alatt a memóriaigény és a futásidő csökkentését értjük.

- **Redukció és dekompozíció különbségei és hasonlóságai?**

Probléma redukciónál halmazokat állítunk elő, amik állapotokat tartalmaznak. Redukciós operátorokat keresünk, amelyek egy adott **B** állapot halmazhoz azon állapotok **A** halmazát rendeli, amelynek bármely *a* állapotából az **M** a **B** halmaz valamelyik *b* állapotába vezet.

A végalmaz *többnyire* minden végállapotot tartalmaz.

A kezdőalmaz *többnyire* csak kezdőállapotokat tartalmaz.

A cél olyan redukciós operátor sorozat megtalálása, ami a végalmazból a kezdőalmazba vezet, a megoldás ezen redukciós operátorokhoz tartozó műveletek fordított sorrendű kiolvasása lesz.

Probléma dekompozíciónál egy adott problémát részproblémákra bontunk, és ezt addig ismétljük, amíg atomi problémákhoz nem érünk. *Hasonlóan a redukcióhoz*, operátorokat keresünk, de ezek nem halmazok közti hozzárendeléseket, hanem dekomponálást fognak végezni. A megoldás ebben az esetben sokszor nem egyértelmű. A dekomponálás előre, az operátorok műveleteinek elvégzése hátra felé történik.

- **Minimax módosításainak felsorolása. Min javítanak ezek?**

Kiértékelő függvény:

- *Átlagoló kiértékelés:* a kiértékelő függvény esetleges tévedéseinek kisimítása
- *Váltakozó mélységű kiértékelés:* a kiértékelő függvény minden ágon a reális értéket mutassa
- *Szelektív kiértékelés:* a memóriaigényt csökkenti

Változatok:

- *Negamax algoritmus:* könnyebb implementálni
- *Alfa-béta algoritmus:* kisebb a memóriaigénye, mert csak 1 utat tárol és a vágások miatt jobb a futási ideje is

- **Szélességi és mélységi korlátos mélységi bejárás összehasonlítása.**

- *Mélységi korlátos mélységi bejárás:*
Végtelen gráfokban biztosan talál megoldást a mélységi korláton belül
- *Szélességi korlátos mélységi bejárás:*
Végtelen gráfokban nem biztos, hogy talál megoldást, mivel mélységi bejárást csinál

- **Mi a nem-determinisztikus rezolúció?**

A rezolúció nem determinisztikus. Egy lépésben:

- Egyszerre több rezolváló klóz pár is lehet
- Egy klóz párban több komplement literál is lehet
- Ugyanannak a literálnak több előfordulása is lehet

- **Visszafelé haladó szabályalapú logikai reprezentáció, formai megszorításai.**

WTF

Gyakorlati rész

1. **Állapottér reprezentáció: 3 kocka, 2 golyó, 1 robotkar, ami golyót mozgat dobozok között. Kezdetben X-ben A, Y-ban B golyó van, Z üres. Cél: X és Y tartalmának cseréje.**
2. **Adott gráfon visszalépéses keresés. (Sorszámozni az éppen aktuális csúcsot [egy csúcsnál több sorszám is lehet], X-szel jelölni azokat a csúcsokat, ahonnan visszalép, megadni a visszalépések számát és a végeredmény utat.**
3. **Adott ÉS/VAGY gráfból közönséges irányított gráfot készíteni.**

2016.06.22.

Elméleti rész

- **Összehasonlítani a lokális keresést, a visszalépéses keresést és a gráfkeresés globális munkaterület, keresési szabály és vezérlési stratégia alapján.**

Globális munkaterület:

- *Lokális keresés:* egy csúcs és annak a szűk környezete
- *Visszalépéses keresés:* egy út a startcsúcsból az aktuális csúcsba és az arról leágazó, még ki nem próbált élek
- *Gráfkeresés:* egy olyan részgráf, a startcsúcsból induló, már feltárt út

Keresési szabály:

- *Lokális keresés:* az aktuális csúcsot minden lépésben egy, az annak környezetében lévő, "jobb" csúccsal cseréljük le
- *Visszalépéses keresés:* a nyilvántartott út végéhez egy új, még ki nem próbált él hozzáfűzése, vagy a legutolsó él törlése (visszalépés)
- *Gráfkeresés:* az egyik útvégi csúcs kiterjesztése

Vezérlési stratégia:

- *Lokális keresés:* a "jobbság" eldöntésére kiértékelő függvényt használ, ami remélhetőleg annál jobb értéket ad egy csúcsra, minél közelebb van a célhoz
- *Visszalépéses keresés:* a visszalépés szabályát csak a legvégső esetben alkalmazza. Visszalépés feltételei: zsákutca, zsákutca torkolat, kör, mélységi korlát
- *Gráfkeresés:* Mindig a legkedvezőbb csúcs kiterjesztésére törekszik
- **ÉS/VAGY gráf**
Olyan $R=(N,A)$ gráf, ahol:
 - N : a csúcs egy részproblémát jelöl
 s startcsúcs: $s \in N$
 t célcsúcsok: $t \in T \subseteq N$
 - A : élköteg, ami a dekomponálendő probléma csúcsából dekomponálással kapott részproblémák csúcsaiba vezet. Élköteg élei közötti kapcsolatok fajtái:
 - *"ÉS"*: a probléma megoldásához annak minden részproblémáját meg kell oldani
 - *"VAGY"*: választhatunk, hogy melyik élköteg mentén oldjuk meg a problémát
- **Optimális útkeresési problémákkal kapcsolatban tanult algoritmusok felírása.**

A nem-informált algoritmusok közül

Megnevezés	Definíció	Állítás
Szélességi bejárás	$f=g,$ $c(n,m)=1$	Optimális (legrövidebb) megoldást adja, ha van
Egyenletes bejárás	$f=g$	Optimális (legolcsóbb) megoldást adja, ha van és egy csúcsot legfeljebb egyszer terjeszt ki

A heurisztikus algoritmusok közül

Megnevezés	Definíció	Állítás
A* algoritmus	$f=g+h, h \leq h^*, h \geq 0$	Optimális megoldást ad, ha van, még végtelen gráfokban is
A ^c algoritmus	$f=g+h, h \leq h^*, h \geq 0, h(n)-$ $h(m) \leq c(n,m)$	Optimális megoldást ad, ha van és ugyanazt a csúcsot nem terjeszti ki kétszer

- Hegymászó algoritmus és tabu keresés közti különbség**

A hegymászó algoritmus lokális optimum hely körül, illetve ekvidisztans felületen lévő körön végtelen működésbe eshet, ezt hivatott kiküszöbölni a tabu keresés a memória növelése által.

A tabu keresés nyilvántart egy tabu halmazt és az optimális csúcsot.

Mindig az aktuális csúcs legjobb, de nem a tabu halmazban lévő gyerekére lép. Ha az aktuális csúcs jobb, mint a jelenlegi optimális csúcs, akkor kicseréljük az optimális csúcsot az aktuális csúcsra. Az algoritmus minden lépésben frissíti a sorszerkezetű tabu halmazt az aktuális csúccsal.

Az algoritmus terminál, ha az optimális csúcs célcúcs, vagy ha az optimális csúcs már régóta nem változott.

MI feladatsorok kidolgozásai

2009-05-27

1. Milyen formában jelennek meg az általános keresőrendszer komponensei egy gráfkereső algoritmus esetén?

1. keresőgráf (G) : a reprezentációs gráf eddig bejárt és eltárolt része
2. nyílt csúcsok halmaza (NYÍLT) : kiterjesztésre várakozó csúcsok, amelyeknek gyerekeit még nem vagy nem eléggé jól ismerjük
3. kiterjesztett csúcsok halmaza (ZÁRT) : azok a csúcsok, amelyeknek a gyerekeit már előállítottuk
4. kiterjesztés (Γ) : egy csúcs összes gyermekének előállítása a hozzájuk vezető élekkel együtt
5. kiértékelő függvény (f : NYÍLT \rightarrow R) : kiválasztja a megfelelő nyílt csúcsot kiterjesztésre

2. Visszalépéses keresés első változatának algoritmus

```
Recursive procedure VL1(akt) return megoldás
    if cél(akt) then return(nil) endif
    for  $\forall \text{új} \in \Gamma(\text{akt})$  loop
        megoldás := VL1(új)
        if megoldás != hiba then
            return fűz((akt,új),megoldás) endif
    endloop
    return(hiba)
end
```

3. Általános keresőrendszer komponensei

A produkciós rendszer (keresőrendszer) három összetevője:

1. Globális adatbázis
A globális adatbázis a feladat reprezentációs gráfjának a megoldása során előállított részét tartalmazza.
2. produkciós szabályok
A produkciós szabályok a globális adatbázison értelmezett operátorok. Egy produkciós szabály az alkalmazása során módosítja az adatbázis tartalmát.
3. vezérlési stratégia
A vezérlési stratégia egy elsőbbségi sorrendet állít fel az alkalmazható produkciós szabályok között.

4. Perceptron modell

[TODO]

5. Kétszemélyes játék állapotér reprezentációja

(Tic Tac Toe?)

Állapotok halmaza:

Az állapotokban le kell tárolni a lépésben lévő játékost és a táblát. Egy mező értéke akkor nulla, ha oda még egyik játékos se tette a jelét, különben a játékoshoz rendelt számát tartalmazza.

$$A = \{ (T_z^* z, p) \mid z \in N, \forall i,j T_{i,j} \in \{0, 1, 2\}, p \in \{1, 2\} \}$$

kezdőállapot:

Kezdőállapotban a teljes tábla üres és az 1-es játékos kezd.

$$K = \{ [[0 \dots 0] \dots [0 \dots 0]], 1 \}$$

célállapot:

Két okból érhet véget a játék:

1. az egyik játékosnak kigyűlik egymás mellett az 5 jel.
2. betelik a tábla

Művelethalmaz:

Művelet, hogy a tábla egy mezőjére rakja le a játékos jelét.

$$M = \{ \text{lerak}_{x,y} \mid 0 \leq x, y < z, z \in N \}$$

Művelet előfeltétele, hogy az x,y pozícion található helyen a játék folyamán eddig egy játékos se helyezett el jelet, tehát $T_{x,y} = 0$.

6. Milyen hiányosságok vannak a klasszikus logikai következtetésben az emberi gondolkodáshoz képest?

[TODO]

7. Probléma-dekompozíciós eljárás reprezentációja

Probléma-dekompozíció reprezentációjának nevezzük a probléma részekre bontásának szeléletére épülő feladat reprezentációt. Ez, a probléma illetve a részproblémák leírása mellett, a közöttük kapcsolatot teremtő úgynevezett redukciós operátorok megadását is jelenti. Probléma-dekompozíció reprezentációjának szemléltetésére is gráfokat használunk, a megfogalmazott feladatok egy ÉS/VAGY gráfon történő keresési problémaként értelmezhetők.

A reprezentációhoz meg kell adnunk:

1. a feladat részproblémáinak általános leírását,
2. az eredeti problémát,
3. az egyszerű problémákat, amelyekről könnyen eldönthető, hogy megoldhatók-e vagy sem, és a dekomponáló műveleteket:

$$D: \text{probléma} \rightarrow \text{probléma}^+ \text{ és } D(p) = \langle p_1, \dots, p_n \rangle$$

8. Definiálja a helyettesítést és az egyesítő helyettesítést!

[TODO]

9. Milyen okai vannak a rezolúciós eljárás nemdeterminisztikusságának?

A rezolúció nem-determinisztikus, ha egy lépésben

1. több klóz párt is rezolválhatunk
2. egy klóz párban több komplementer literál pár is lehet
3. ugyanannak a literálnak több előfordulása lehet

$\{P(x, f(a))P(x, f(y))Q(y), \neg P(z, f(a))\neg Q(z), P(u, f(a))\neg Q(a)\}$

10. Hogyan készíthető gráfrepresentáció az előre és visszafelé haladó szabályalapú reprezentációhoz? (Tk. 34 oldal)

1. Előre:

A kezdőállapotból elindulva fokozatosan építi fel a célállapotba vezető műveletsorozatot, közben esetleg sok felesleges műveletet is kipróbál. Az ilyen irányba működő produkciós rendszereket előrehaladó vagy adatvezérelt produkciós rendszereknek nevezzük.

2. Visszafelé:

Amennyiben a feladat egyetlen és ismert célállapottal rendelkezik, akkor alkalmazhatunk úgynevezett visszafelé haladó vagy célvezérelt produkciós rendszert is. A visszafelé haladó produkciós rendszer globális adatbázisát kezdetben a célállapot tartalmazza és a produkciós szabályok inverzeit alkalmazza egészen addig amíg a kezdőállapotot el nem éri.

#####

2009-06-03

1. Adja meg az általános keresorendszer algoritmusát! (TK. 29. oldal)

```
Procedure KR
  ADAT := kezdeti érték
  while !terminálási feltétel(ADAT) loop
    SELECT SZ FROM alkalmazható szabályok
    ADAT := SZ(ADAT)
  endloop
end
```

2. Milyen feltételek mellett terminál, és talál megoldást a visszalépéses keresés első változata? (TK. 41. oldal)

A visszalépéses keresés akkor ér véget, ha talál egy célcsúcsba vezető utat, vagy már minden lehetséges utat végignézett. Ebből látszik, hogy a keresés befejeződhet sikeresen és sikertelenül. Véges körmentes irányított gráfokon a VL1 mindig terminál, és ha létezik megoldás, akkor talál egyet.

Visszalépés feltételei VL1 nél:

zsákutca: az aktuális csúcsból (azaz az aktuális út végpontjából) nem vezet tovább él
zsákutca torkolat: az aktuális csúcsból kivezető utak nem vezetnek célba

3. Milyen feltételek mellett talál megoldást az általános gráfkereso algoritmus?

A GK véges delta-gráfban mindig terminál.

Ha egy véges delta-gráfban létezik megoldás, akkor a GK megoldás megtalálásával terminál.

(5.grafker.pdf - 2. oldal)

4. Milyen okok akadályozzák egy probléma dekompozíciós reprezentációjának megfogalmazását?

Dekomponáló műveleteket nagyon nehéz megtalálni.

- Nem minden feladat dekomponálható.
- Könnyen kaphatunk hamis dekomponáló műveleteket.

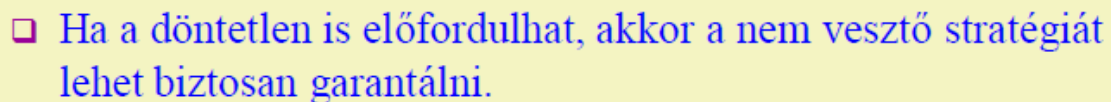
Az egyszerű probléma felismerése sem mindig egyértelmű:

$$\int \sin(x)e^x dx = \dots = \sin(x)e^x - \cos(x)e^x - \int \sin(x)e^x dx$$

A megoldás kiolvasása sem nyilvánvaló.

(6.2.dekomp.pdf - 1. oldal)

- Ha a döntetlen nem megengedett, akkor bármelyik kétszemélyes játékban az egyik játékos számára biztosan létezik nyerő stratégia.



6. írja fel az evolúciós algoritmus általános sémáját!

endloop

7. Melyek klóz-formára hozás lépései az elsőrendű predikátumkalkulusban?

1. Kiküszöböljük az \leftrightarrow és a \rightarrow műveleti jeleket.
2. Redukáljuk a negációk hatáskörét.
3. Standardizáljuk a változókat (kvantoronkénti átnevezés).
4. Egzisztenciális kvantorok Skolemizálása.
5. Univerzális kvantorok kiemelése a sorrendjük megtartásával.
6. A formula többi részét konjunktív normálformára alakítjuk.

7. Kialakítjuk a klózok halmazát (a kvantorokat és a konjunkciós műveleti jeleket elhagyjuk, a változókat klózonként egyedivé nevezzük át.)
(10.auto_kovetkeztet.pdf - 2. oldal)

8. Adja meg a támasztó(támogató) halmaz stratégia és a lineáris input stratégia meghatározását. Teljesek-e ezek a stratégiák?

1. Támogató-halmaz stratégia: egyik szülő az S kiinduló klózoknak egy adott T részhalmazából lett levezetve. (Teljes, ha S-T kielégíthető)
 2. Lineáris input stratégia: egyik szülő az előző lépésben kapott klóz (kivéve az első lépést), a másik egy kiinduló klóz. (Teljes Horn klózok esetén)
- (10.auto_kovetkeztet.pdf - 4. oldal)

9. Jellemezze a visszafele haladó szabályalapú reprezentáció formuláinak alakját!

□ **Tény:**

- $L_1 \wedge \dots \wedge L_n$ alakú univerzálisan kötött kifejezés, ahol L_1, \dots, L_n literálok.

□ **Szabályok:**

- $W \rightarrow L$ alakú univerzálisan kötött kifejezések, ahol L egy literál, a W pedig ÉVF kifejezés

□ **Cél:**

- egzisztenciálisan kötött **tetszőleges** ÉVF kifejezés

ÉS/VAGY formájú (ÉVF) kifejezések:

- literálok
- $A \wedge B, A \vee B$ alakú formulák, ahol az A és B is ÉVF kifejezés.

(10.auto_kovetkeztet.pdf - 7. oldal)

10. Ismertesse, hogy milyen megoldásokat tanult a nem-monoton következtetés területén a hiányos axióma rendszer illetve az ellentmondásos axióma rendszer kezelésére!

What?

2010-06-04

1. Általános keresőrendszer algoritmusa

```
ADAT := kezdeti érték
while !terminálási feltétel(ADAT) loop
    SELECT SZ FROM alkalmazható szabályok
    ADAT := SZ(ADAT)
endloop
```

[gt 1. dia, 3. dia]

3. biz. be, hogy A* algoritmus monoton heurisztikával minden csúcsot csak egyszer terjeszt ki!

Nem találtam

4. hiperút fogalma

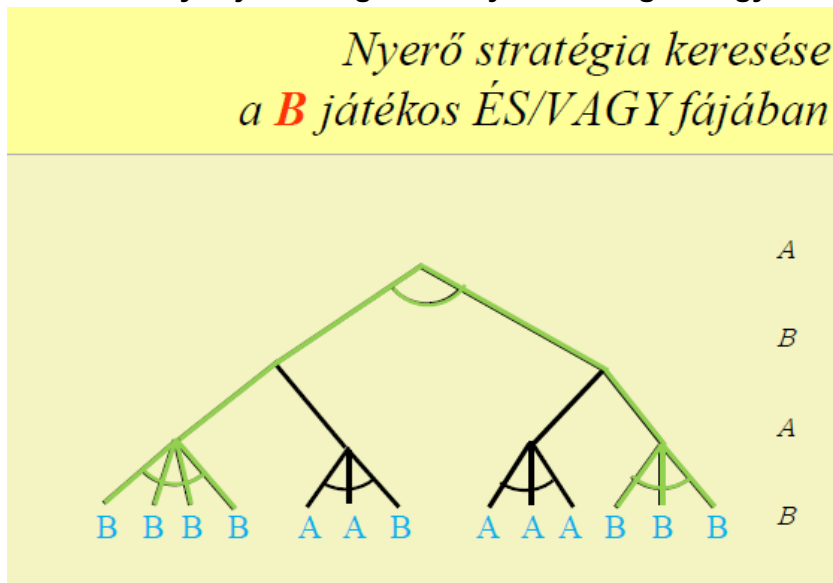
A hiperút egy olyan véges részgráf, amelyben

- M csúcsaiból nem indul hiper-él,
- többi csúcsból egyetlen hiper-él indul,
- bármelyik csúcs elérhető az n csúcsból egy közöséges irányított úton.

Hiperút hossza a benne levő élek száma.

(6.2.dekomp.pdf - 4. oldal)

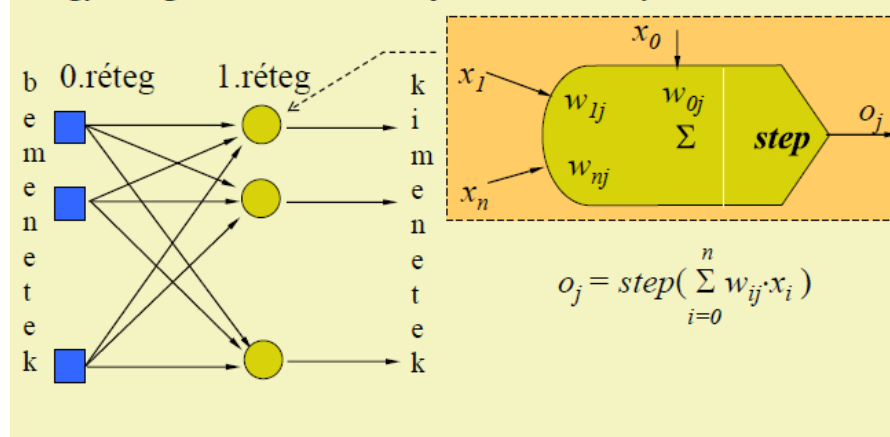
5. kétszemélyes játéknál gráfban nyerő stratégiát hogyan kell ábrázolni?



(7.jatek.pdf - 2. oldal)

6. perceptron modell - elemei, topológia, tanulási módszer

□ Rosenblatt (lépcsős aktivizációs függvényű) perceptronok egy rétegű előrecsatolt hálójá, delta-szabály tanulással.



Elemei:

Bemenetek, Rétegek, Kimenetek.

Tanulás:

Általánosított perceptronok esetén a neuron w súlyait tanuljuk meg, amely nemcsak a neuron számítási képletre hat ki, hanem a hálózat topológiájára is.

A minta a neuron egy súlyát a $w := w + \Delta w$ alapján módosítja.

A Δw kiszámítása a neuron számított kimenetére támaszkodik, ehhez pedig a közvetlen megelőző neuronok számított kimeneteit is szükségesek.

- felügyelt tanulás esetén a neuron (összetett esetben a neuront tartalmazó hálózat) várt kimenetét is használjuk,
- felügyelet nélküli tanulás esetén a várt kimenetet nem kell ismerni.

Egy egyrétegű perceptron modell csak feltételeket képes felismerni (osztályozni), de egy kétrétegű már ezek kombinációit, azaz konvex poliédereket is, egy három rétegű pedig tetszőleges poliédereket.

(8.neuron.pdf - 2.,3. oldal)

7. lineáris input, őstre korlátozott rezolúciós stratégia

Lineáris input stratégia: egyik szülő az előző lépésben kapott klóz (kivéve az első lépést), a másik egy kiinduló klóz. (Teljes Horn klózok esetén)

Őstre korlátozott stratégia: egy szülő az előző lépésben kapott klóz, a másik vagy egy kiinduló klóz vagy egy őse az első szülőnek. (Teljes)

(10.auto_kovetkeztet.pdf - 4. oldal)

8. visszafelé szabályalapú levezetésnek a formulái milyen alakúak?

□ Tény:

- $L_1 \wedge \dots \wedge L_n$ alakú univerzálisan kötött kifejezés, ahol L_1, \dots, L_n literálok.

□ Szabályok:

- $W \rightarrow L$ alakú univerzálisan kötött kifejezések, ahol L egy literál, a W pedig ÉVF kifejezés

□ Cél:

- egzisztenciálisan kötött tetszőleges ÉVF kifejezés

ÉS/VAGY formájú (ÉVF) kifejezések:

- literálok
- $A \wedge B, A \vee B$ alakú formulák, ahol az A és B is ÉVF kifejezés.

(10.auto_kovetkeztet.pdf - 7. oldal)

9. helyettesítés, egyesítő helyettesítés

10. valami hogy az alternatív bigyónál milyen módszerekkel lehet feloldani az ellentmondásos dolgot, meg egy másik...

2010-06-09

1. egy probléma állapotterét hogyan valósítjuk meg gráfrepresentációval?

Állapot-gráf (az állapotter-reprezentáció reprezentációs gráfja):

- állapot csúcs
- művelet hatása irányított él
- művelet költsége élköltség

- kezdő állapot startcsúcs
- célállapotok célcsúcsok
- műveletsorozat hatása irányított út

(2.allrepr.pdf - 1. oldal)

2. a visszalépéses keresés 2-es változata milyen gráfban terminál és mikor megoldással

A visszalépéses algoritmus második változata az, amikor a visszalépés feltételei közül mindet beépítjük a kereső rendszerbe.

A VL2 ?-gráfban mindig terminál. Ha létezik a mélységi korlátnál nem hosszabb megoldás, akkor megtalál egy megoldást.

(4.visszalep.pdf - 4. oldal)

3. milyen nem informált gráfkereséseket ismer. definiálja ezeket feltételezve, hogy ismerjük a gráfkeresési alg-t.

Mélységi:

- $f = -g$, $c(n,m) = 1$,
- végtelen gráfokban csak mélységi korláttal garantál megoldást

Szélességi:

- $f = g$, $c(n,m) = 1$,
- optimális (legrövidebb) megoldást adja, ha van,
- egy csúcsot legfeljebb egyszer terjeszt ki

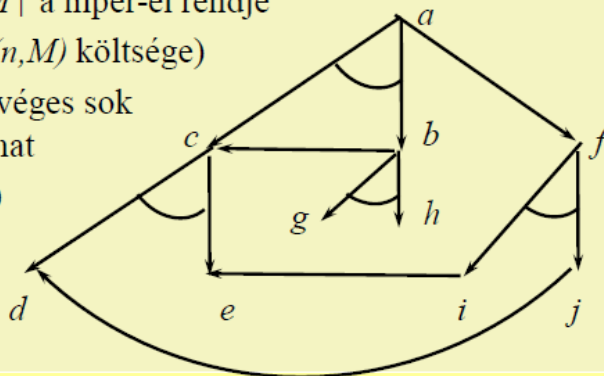
Egyenletes:

- $f = g$,
- optimális (legolcsóbb) megoldást adja, ha van,
- egy csúcsot legfeljebb egyszer terjeszt ki.

(5.grafker.pdf - 3. oldal)

4. definiálja az ÉSVAGY gráfot

1. Az $R=(N,A)$ élsúlyozott irányított hiper-gráf, ahol az
 - N a csúcsok halmaza,
 - $A \subseteq \{ (n,M) \in N \times N^+ \mid 0 \neq |M| < \infty \}$ a hiper-élek halmaza, $|M|$ a hiper-él rendje
 - $(c(n,M))$ az (n,M) költsége
2. Egy csúcsból véges sok hiper-él indulhat
3. $(0 < \delta \leq c(n,M))$



(6.2.dekomp.pdf - 4. oldal)

5. ismertesse a minimax algoritmust

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 := $\max(\text{gyerek}_1, \dots, \text{gyerek}_k)$
 - Az ellenfél (MIN) csúcsaihoz azok gyermekeinek minimumát: szülő := $\min(\text{gyerek}_1, \dots, \text{gyerek}_k)$
4. Soron következő lépésünk ahhoz az álláshoz vezet, ahonnan a gyökérhez felkerült a legnagyobb érték.

(7.jatek.pdf - 3. oldal)

6. milyen szempontokat kell figyelembe venni az evolúciós algoritmus kiválasztó operátorának (szelekció?) konstruálásakor?

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
- Rangsorolós: rátermetség alapján sorba rendezett egyedek közül a kisebb sorszámúakat nagyobb valószínűséggel választja ki
- Versengő: véletlenül kiválasztott egyedcsoportok (pl. párok) legjobb egyedét választja ki.
- Csonkolósos v. selejtezős: a rátermetség szerint legjobb (adott küszöbérték feletti) valahány egyedből véletlenszerűen választ néhányat.

(9.evolucio.pdf - 4. oldal **Nem vagyok biztos benne, hogy ez jó válasz**)

7. erre nem emlékszem rendesen de talán: írja le az egyesítést

What?

8. ismertesse az egységklóz / támasztóhalmaz stratégiát. teljesek-e ezek?

Egységklóz stratégia: az egyik szülő egységklóz (egy literálból álló klóz). (Teljes Horn klózik esetén)

Támogató-halmaz stratégia: egyik szülő az S kiinduló klóziknak egy adott T részalmazából lett levezetve. (Teljes, ha S-T kielégíthető)

2011-05-24

1. Nevezze meg és jellemezze a kereső rendszerek fő részeit!

A globális munkaterületen a reprezentációs gráf egy részgráfja jelenik meg.

- vagy csak egy csúcs, esetleg annak közvetlen szomszédjai
- vagy a startcsúcsból kivezető egyik út kezdőszakasza
- vagy a startcsúcsból kivezető összes út kezdőszakasza

A keresési szabályok ezt a globális munkaterületen tárolt részgráfot módosítják

- lecserélnek egy csúcsot valamelyik szomszédjára
- hozzávesznek egy tárolt út végéhez egy onnan kiinduló élt
- törölnek egy tárolt út végétől egy élt

A vezérlési stratégia a fenti szabályokból válogat.

(1.bevezetes.pdf - 3. oldal)

2. Melyek a visszalépés feltételei a visszalépéses algoritmus legáltalánosabb változatában?

- zsákutca: az aktuális csúcsból (azaz az aktuális út végpontjából) nem vezet tovább él
- zsákutca torkolat: az aktuális csúcsból kivezető utak nem vezetnek célba

(4.visszalep.pdf - 1. oldal)

3. Mikor nevezünk egy gráfkereső algoritmust A* algoritmusnak, és melyek ennek legfőbb jellemzői?

Nem találom

4. Mit jelent kétszemélyes játékoknál a nyerő stratégia, és milyen állítást mondtunk ki ezzel kapcsolatban?

Egy játékos nyerő stratégiája egy olyan elv, amelyet betartva az ellenfél minden lépésére tud olyan választ adni, hogy megnyerje a játékot.

A nyerő stratégia NEM egyetlen győztes játszma, hanem olyan győztes játszmák összessége, amelyek közül az egyiket biztos végig tudja játszani az a játékos, aki rendelkezik a nyerő stratégiával.

A nyerő stratégia keresése egy ÉS/VAGY gráfbeli hiper-út keresési probléma.

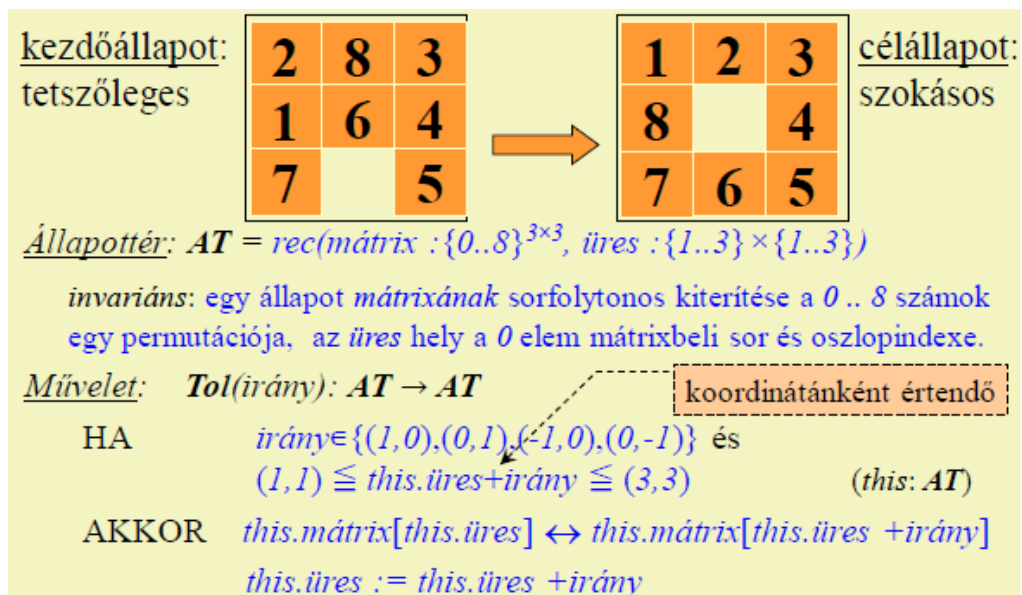
(7.jatek.pdf - 2. oldal)

5. Mi történik (milyen lépések hajtnak végre) egy evolúciós algoritmus egy iterációja során?

- **Szelekció:** Kiválasztunk néhány (lehetőleg rátermett) egyedet szülőnek.
- **Rekombináció** (keresztelés): Szülőkből utódok készülnek úgy, hogy a szülők tulajdonságait örököljék az utódok.
- **Mutáció:** Az utódok tulajdonságait kismértékben módosítjuk.
- **Visszahelyezés:** Új populációt alakítunk ki az utódokból és a régi populációból.

(9.evolucio.pdf - 1. oldal)

I. Adjon állapottér-reprezentációt a 8-as kirakó játékra! (Egy 3x3-as keretben 8 darab 1x1-es méretű 1-től 8-ig megszámozott lapocska található és egy üres hely. Egy lapocskát a kereten belül a vele szomszédos üres helyre el lehet tolni. Egy megadott kezdőállásból elindulva rendezzük át a lapocskákat úgy, hogy középen legyen az üres hely, a lapocskák pedig az óra járásával megegyező irányban legyenek megszámozva a bal felső saroktól indulva.)



(2.allrepr.pdf - 3. oldal)

II. "Aki sokat iszik, annak remeg a keze. Akinek remeg a keze, az sok italt kilötyköl a poharából. Aki sok italt kilötyköl a poharából, az keveset iszik. Lássuk be, hogy aki sokat iszik, az keveset iszik!"

Formalizálja a fenti mondatokat, adjon szabályalapú reprezentációt erre a problémára, majd alakítsa át klóz alapú reprezentációra, és oldja meg rezolúcióval!

TODO

2011-05-26

1, vezérlési stratégiák osztályzása

Elsődleges:

- független a feladattól és annak modelljétől: nem merít sem a feladat ismereteiből, sem a modell sajátosságaiból.
- 1. nemmódosítható (lokális keresések, evolúciós algoritmus, rezolúció)
- 2. módosítható (visszalépéses keresések, gráfkeresések)

Másodlagos:

- nem függ a feladat ismereteitől, de épít a feladat modelljének általános elemeire.

Heurisztika:

- a feladattól származó, annak modelljében nem rögzített, de a megoldást segítő speciális ismeret.

(3.lokalker.pdf - 1. oldal)

2, visszalépéses keresés előnyei/hátrányai

ELŐNYÖK:

- mindig terminál, talál megoldást
- könnyen implementálható
- kicsi memória igény

HÁTRÁNYOK:

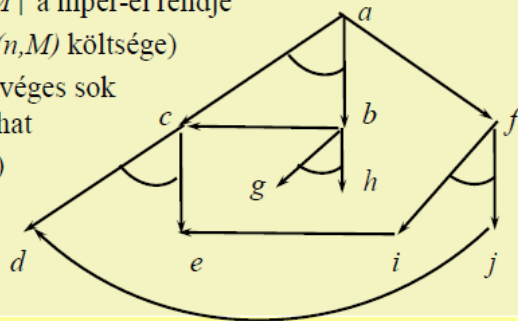
- nem ad optimális megoldást. (iterációba szervezhető)

- kezdetben hozott rossz döntést csak sok visszalépés korrigál (visszaugrások keresés)
- egy zsákutca részt többször is bejárhat a keresés
(4.visszalep.pdf - 5. oldal)

3, A^* milyen gráfkereső és mi jellemző rá (vagy valami hasonló)

TODO

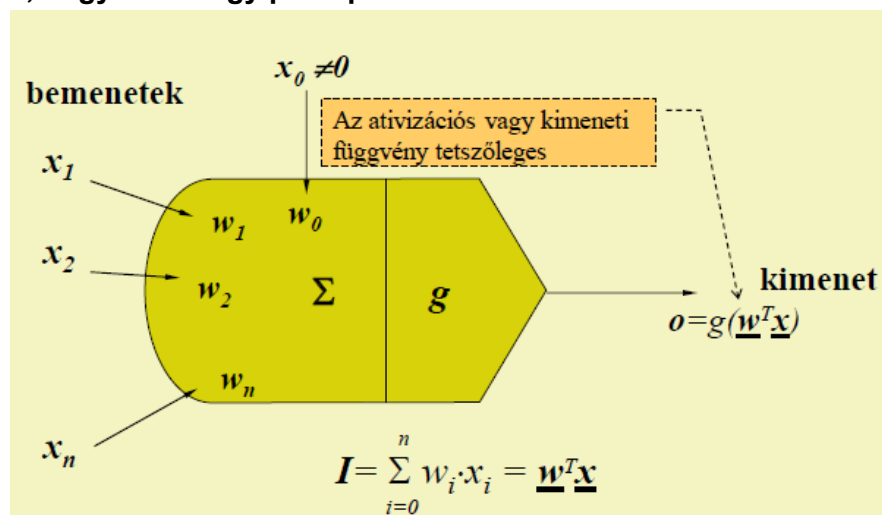
1. Az $R=(N,A)$ élsúlyozott irányított hiper-gráf, ahol az
 - N a csúcsok halmaza,
 - $A \subseteq \{ (n,M) \in N \times N^+ \mid 0 \neq |M| < \infty \}$ a hiper-élek halmaza, $|M|$ a hiper-él rendje
 - $(c(n,M))$ az (n,M) költsége
2. Egy csúcsból véges sok hiper-él indulhat
3. $(0 < \delta \leq c(n,M))$



4, ÉS/VAGY gráf def

(6.2.dekomp.pdf - 4. oldal)

5, hogy néz ki egy perceptron



(8.neuron.pdf - 1. oldal)

F1 n-királynő probléma reprezentációja



(2.allrepr.pdf - 2. oldal)

F2 visszalépéses keresésen alapuló gráfos feladat

2011-05-31

1) Jellemeze a problémás és a tabu keresés által adott megoldást.

Hegymászó algoritmus a lokális optimum hely körül vagy ekvidisztans felületeten (azonos értékű szomszédos csúcsok között) található körön végtelen működésbe eshet.

Ennek a hátránynak a kiküszöbölésére megoldást nyújt a tabu keresés, mely aktuális csúcson kívül (akt) nyilvántartja az utolsó néhány érintett csúcsot (tabu halmaz), és a legjobb csúcsot (opt).

Az algoritmus minden lépésben az aktuális csúcs egy gyerekére lép, kivéve a Tabu halmazban lévőket, ha akt jobb, mint az opt, akkor az opt az akt lesz, és frissíti a sorszerkezetű tabu halmazt akt-al. Terminálási feltétel, ha az opt a célcsúcs, illetve ha opt sokáig nem változik.

Algoritmus:

```

    akt, opt, Tabu := start, start, ∅
    while not (opt ∈ T ∨ opt régebb óta nem változott) loop
        akt := optf(Γ(akt) - Tabu)
        Tabu := Módosít(akt, Tabu)
        if f(akt) jobb, mint f(opt) then opt := akt
    endloop

```

A törzs első sora helyettesíthető a következővel bővebben:

```

    if Γ(akt) = ∅ then return nem talált megoldást
    else if Γ(akt) - Tabu = ∅ then akt := optf(Γ(akt)) else akt := optf(Γ(akt) - Tabu)

```

Az algoritmus előnye, hogy a tabu méreténél rövidebb köröket észleli.

Hátránya, hogy a Tabu halmaz méretét kísérletezéssel kell belőni, és zsákutcába futva beragad.

[gt 3. dia]

2) visszalépéses keresés globális munkaterülete, keresési szabálya, vezérlési stratégiája.

Globális munkaterülete:

- egy út a startcsúcsból az aktuális csúcsba (ezen kívül az útról leágazó még ki nem próbált élek)
 - Kezdetben a startcsúcsot tartalmazó nulla hosszúságú út
 - Terminálás célcsúccsal vagy startcsúcsból való visszalépéssel

Keresés szabályai:

- a nyilvántartott út végéhez egy új (ki nem próbált) él hozzáfűzése, vagy a legutolsó él törlése (visszalépés szabálya)

Vezérlés stratégiája a visszalépés szabályát csak a legvégső esetben alkalmazza.

(4.visszalep.pdf - 1. oldal)

3) Milyen eredményeket mondtunk ki az általános gráfkereső algoritmusról?

Bebizonyítható:

- A GK ?-gráfban a működése során egy csúcsot legfeljebb véges sokszor terjeszt ki. (tehát a körökre nem érzékeny)
- A GK véges ?-gráfban mindig terminál.
- Ha egy véges ?-gráfban létezik megoldás, akkor a GK megoldás megtalálásával terminál.

(5.grafker.pdf - 2. oldal)

5) Milyen a jó szelekciós operátor? Adjon egy példát rá!

Nem találom

F1) Adjon állapotér-reprezentációt kielégíthetőségi problémára: Adott egy n darab ítéleváltozó (logikai értékű változó) segítségével felírt nulladrendű logikai formula. A változók milyen értéke mellett lesz a formula igaz? Próbálja megbecsülni egyfelől az ön által adott állapotér méretét, másfelől a problémater méretét (a műveletek által kijelölt startcsúcsból kiinduló lehetséges utak száma)!

TODO

F2) egy megadott fiktív játéka kiértékelése minimax algoritmussal, mi a soron következő játékos lépése

TODO

2011-06-03

1. Milyen hatással van a heurisztika általában a kereső rendszerek működésére?

A heurisztika olyan, a feladathoz kapcsolódó ötlet, amelyet közvetlenül építünk be egy algoritmusba azért, hogy annak eredményessége és hatékonysága javuljon, habár erre általában semmiféle garanciát sem ad.

(3.lokalker.pdf - 3. oldal)

2. Írja le a hegymászó algoritmust!

```

akt := start
while akt  $\notin$  T loop
    akt :=  $\text{opt}_f(\Gamma(\text{akt}) - \pi(\text{akt}))$ 
endloop

```

while törzse a következővel helyettesíthető bővebben:

```

    if  $\Gamma(\text{akt}) = \emptyset$  then return nem talált megoldást
    if  $\Gamma(\text{akt}) - \pi(\text{akt}) = \emptyset$  then akt :=  $\pi(\text{akt})$  else akt :=  $\text{opt}_f(\Gamma(\text{akt}) - \pi(\text{akt}))$ 

```

[gt. 3. dia]

3. Mit tesz az általános gráfkereső algoritmus akkor, amikor egy már korábban felfedezett csúcshoz talál minden addiginál olcsóbb utat?

Ekkor annak a korábbi csúcshoz a szülője az aktuális kiterjesztés gyökere lesz (tudom, nem valami érthető), az útköltség pedig a szülő és a startcsúcs közti útköltség + a szülő és az aktuális(korábbi) csúcs útköltsége.

Formálisan:

Ha m régi csúcs, amelyhez olcsóbb utat találtunk,
 azaz $m \in G$ és $g(n) + c(n, m) < g(m)$ akkor
 $g(m) := g(n) + c(n, m)$.

(5.grafker.pdf - 1. oldal)

4. Mit tartalmaz egy probléma dekompozíciós reprezentációja?

A reprezentációhoz meg kell adnunk:

- a feladat részproblémáinak általános leírását,
- az eredeti problémát,
- az egyszerű problémákat, amelyekről könnyen eldönthető, hogy megoldhatók-e vagy sem, és
- a dekomponáló műveleteket:

○ D : probléma \rightarrow probléma⁺ és $D(p) = \langle p_1, \dots, p_n \rangle$

(6.2.dekomp.pdf - 1. oldal)

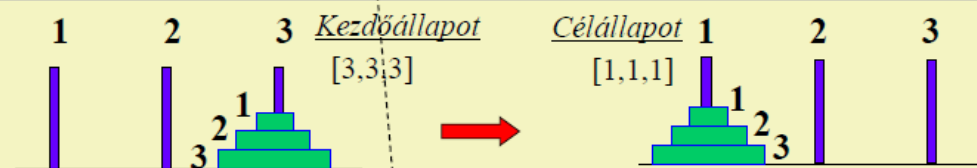
5. Adjon példát legalább három rekombinációs operátorra!

Nem találom

I. Adjon állapottér reprezentációt a Hanoi-tornyai problémára! (stb. stb.)

1..n intervallummal indexelt egydimenziós tömb, amelynek elemei $\{1,2,3\}$ halmazbeliek.

Hanoi tornyai probléma



Állapottér: $AT = \{1,2,3\}^n$

megjegyzés : a tömb i -dik eleme mutatja az i -dik korong rúdjának számát; a korongok a rudakon méretük szerint fentről lefelé növekvő sorban vannak.

Művelet: $Rak(honnan, hova): AT \rightarrow AT$

HA a *honnan* és *hova* létezik és nem azonos, és van korong a *honnan* rúdon, és a *hova* rúd üres vagy annak legfelső korongja nagyobb *honnan* rúd legfelső korongjánál

AKKOR $this[honnan \text{ legfelső korongja}] := hova$ (*this*: AT)

(2.allrepr.pdf - 1. oldal)

II. A boszorkányok mindenkit elbájolnak, akik nem bájolognak, ám a bájolgókat ők sem bájolják el. (Bájolgó az, aki saját magát elbájolja.) Bizonyítsa be ezek alapján - REZOLÚCIÓVAL- Könyves Kálmán királyunk híres mondását: "Boszorkányok pedig nincsenek!" Adja meg a formulákat, utána a kiinduló klózokat, végül a rezolúciós levezetést! Használja a $V(x,y)$ szimbólumot annak jelölésére, hogy x elbájolja y -t, és a $V(x,x)$ -et arra, hogy x bájolog, azaz magát bájolja el.

TODO

2011-06-08

Milyen célból alkalmazhatunk heurisztikákat a visszalépéses keresésben? Adjon rájuk egy-egy példát!

- Sorrendi szabály: sorrendet ad az aktuális út végpontjából kivezető élek (utak) vizsgálatára
- Vágó szabály: megjelöli azokat az aktuális út végpontjából kivezető éleket (utakat), amelyeket nem érdemes megvizsgálni

Sorrendi heurisztikák az n -királynő problémára:

A mezőkhöz rendelt értékek alapján rendezzük sorba a mezőket.

Diagonális: a mezőn áthaladó hosszabb átló hossza.

Vágás: Forward Checking:

FC algoritmus:

Nemcsak akkor lép vissza, ha az aktuális sorban nincs szabad (ütésben nem álló üres) mező, hanem ha a hátralevő üres sorok valamelyikében nincs már szabad mező, azaz $\exists r \in [k+1..n]: Szabad_r = \emptyset$, akkor is visszalép.

(4.visszalep.pdf - 1,2. oldal)

Mikor nevezünk egy gráfkereső algoritmust A algoritmusnak? Milyen állítást mondhatunk ki vele kapcsolatban?

$f = g+h$ és $0 \leq h$

megoldást ad, ha van megoldás (még végtelen gráfokban is)

Hiányos

(5.grafker.pdf - 3. oldal)

Mit értünk egy hiperút bejárásán?

Egy irányított hiper-út bejárása is egy sorozat, de ez csúcsok sorozatainak sorozata, és ez nem egyértelmű.

(6.2.dekomp.pdf - 4. oldal)

I.Állapottér reprezentáció az utazó ügynök problémára (utazó ügynök leírását is megadta)

TODO

II. alfa-béta eljárással kellett kiértékelni egy megadott fát!

2011-06-09

1. Hogyan definiáljuk a Δ -gráfban egy n -ből m -be vezető út költségét? Hol használja ki a def, hogy Δ -gráfban vagyunk?

TODO

2. Mit nevezünk egy probléma állapotter-reprezentációjának?

- Állapottér: a feladat fókuszában álló adat-együttes (objektum) lehetséges értékeinek (állapotainak) halmaza
 - egy állapot többnyire egy összetett szerkezetű érték, amelynek gyakran meg kell felelnie egy invariáns állításnak is.
- Műveletek: állapotból állapotba vezetnek
 - megadásukhoz: előfeltétel és hatás-leírás
 - invariáns tulajdonságot tartó leképezés
- Kezdő állapot(ok) vagy azokat leíró kezdőfeltétel
- Céllálatpot(ok) vagy célfeltétel

(2.allrepr.pdf - 1. oldal)

4. Minimax algoritmus módosításai, melyek ezek, milyen szempontból javítanak a minimax működésén?

Nem találom

(7.jatek.pdf)

5. Mit jelent a lineárisan szeparálhatóság fogalma, hogyan kapcsolódik a mesterséges neuronhálózatok témaköréhez?

Egyetlen perceptron csak olyan bonyolultságú feladatot képes megoldani, ahol az azonos eredményt adó bemenetek két csoportját egyetlen hipersíkkal el lehet választani, azaz lineárisan szeparálhatók.

Azért nincs XOR műveletet megvalósító perceptron, mert ez a probléma nem lineárisan szeparálható.
(8.neuron.pdf - 3. oldal)

I. Egy homogén bináris p relációról tudjuk, h irreflexív (egy x -re sem teljesül xpx) és tranzitív (bármely x,y,z -re ha xpy és ypz akkor xpz) Bizonyítsa be, hogy ekkor a reláció antiszimmetrikus (bármely x,y -ra ha xpy és ypx akkor $x = y$)! Formalizálja az axiómákat és a célállapítást úgy, hogy a xpy relációt az $R(x,y)$ predikátum szimbólummal, az $x=y$ relációt az $E(x,y)$ predik. szimb-al jelöli. Igazolja az állítást rezolúciós következtetéssel.

TODO

II. A algoritmus működésének szemléltetése egy gráf-reprezentációval megadott problémán keresztül.

2011-06-17

1. visszalépéses keresés: első/második változat

```
Recursive procedure  $VLI(akt)$  return megoldás
1.   if  $cél(akt)$  then return(nil) endif
2.   for  $\forall új \in \Gamma(akt)$  loop
3.        $megoldás := VLI(új)$ 
4.       if  $megoldás \neq hiba$  then
5.           return( $fűz((akt, új), megoldás)$ ) endif
6.   endloop
7.   return(hiba)
end
```

Recursive procedure $VL2(út)$ **return** $megoldás$

```
1.    $akt := utolsó\_csúcs(út)$ 
2.   if  $cél(akt)$  then return( $nil$ ) endif
3.   if  $hossza(út) ≥ korlát$  then return( $hiba$ ) endif
4.   if  $akt ∈ maradék(út)$  then return( $hiba$ ) endif
5.   for  $∀ új ∈ Γ(akt) - π(akt)$  loop
6.        $megoldás := VL2(fűz(út, új))$ 
7.       if  $megoldás ≠ hiba$  then
8.           return( $fűz((akt, új), megoldás)$ ) endif
9.   endloop
10.  return( $hiba$ )
end
```

(4.visszalep.pdf - 1,4. oldal)

2. gráfker: szélességi

TODO

3. és/vagy gráfot mikor, miért használjuk

A közönséges irányított gráfok útkereső algoritmusait szeretnénk adaptálni az ÉS/VAGY gráfbeli keresési problémákhoz: megoldás gráf kereséséhez.

A megoldás gráf egy startcsúsból kiinduló hiper-gráf. A startcsúsból kiinduló hiper-gráfokat megadhatjuk a bejárásaikkal, ennél fogva ábrázolhatóak közönséges irányított utakkal.

Ez lehetővé teszi, hogy minden ÉS/VAGY gráfnak megfeleltethessünk egy olyan közönséges $?$ -gráfot, amelynek megoldási útjai az ÉS/VAGY gráfbeli megoldás gráfok bejárásai, és ezeket közönséges útkereső algoritmusokkal megkereshetjük.

(Nem biztos, hogy ez jó válasz)

(6.2.dekomp.pdf - 4. oldal)

4. evolúciós alg: visszahelyezés

A visszahelyezés a populációnak az utódokkal történő frissítése: Kiválasztja (újabb szelekció) 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 egy utód több példányban is bekerülhet
- ha $u > v$, akkor az utódok közül szelektál

(9.evolucio.pdf - 5. oldal)

5. 3 tulajdonság az emberi logikában, amiért nem lehet a klasszikus logikával modellezni.

Nem találom

I. huszár strat állapotrepr stb (mint előzőeknél)

II. negamax

2011-12-20

1. két személyes játékoknál tetel a nyero strategiarol

Egy játékos nyerő stratégiája egy olyan elv, amelyet betartva az ellenfél minden lépésére tud olyan választ adni, hogy megnyerje a játékot.

(7.jatek.pdf - 2. oldal)

2. A* algoritmus

Nem találom

4. evolucios algoritmus lepesei

1. Szelekció: Kiválasztunk néhány (lehetőleg rátermett) egyedet szülőnek.
2. Rekombináció (keresztelés): Szülőkből utódok készülnek úgy, hogy a szülők tulajdonságait örököljék az utódok.
3. Mutáció: Az utódok tulajdonságait kismértékben módosítjuk.
4. Visszahelyezés: Új populációt alakítunk ki az utódokból és a régi populációból.

(9.evolucio.pdf - 1. oldal)

F1 egy megadott grafra a hegymaszo algoritmus, megadott heurisztikaval

F2 utazo ugynok allatpottterrepresentacio

2012-01-03

F1 egy megadott gráf visszalépéses bejárása

F2 állapotér reprezentáció n királynő problémára

2012. 06. 20.

5. Adjon példát legalább három rekombinációs operátorra!

Nem találom

I. Fekete-fehér kirakó állapotér-reprezentációja. Problématér és állapotér méretének becslése.

Allapot: $AT = \text{rec}(v : \{B, W, _ \}^{n+m+1}, ires : [1.. n+m+I])$

Invariánsok: n darab B , m darab W , 1 üres hely, i -es az üres hely indexe

Műveletek: **TolBal**, **TolJobb**, **UgrikBal**, **UgrikJobb**: $AT \rightarrow AT$

TolBal : HA $this.i \neq I$ ($this : AT$)

AKKOR $this.v[this.i] \leftrightarrow this.v[this.i-1]$

$this.i := this.i - 1$

Kész állapot: $[B, \dots, B, W, \dots, W, _]$

Célállapot: $\forall i,j \in [1.. n+m+I], i < j : \neg (this.v[i]=B \wedge this.v[j]=W)$

Első:

- Végtes körmentes irányított gráfokon (itt nem kell ?-gráf) a VL1 mindig terminál, és ha létezik megoldás, akkor talál egyet.

Második:

- A VL2 ?-gráfban mindig terminál. Ha létezik a mélységi korlátnál nem hosszabb megoldás, akkor megtalál egy megoldást. UI: végtes sok adott korlátnál rövidebb startból induló út van.

(4.visszalep.pdf - 1,4. oldal)

2. Mikor nevezünk egy gráfkereső algoritmust szélességi keresésnek? Milyen állítást mondhatunk ki vele kapcsolatban?

Optimális (legrövidebb) megoldást adja, ha van.

Egy csúcsot legfeljebb egyszer terjeszt ki

Állítás?

(5.grafker.pdf - 3. oldal)

3. Sorolja fel, milyen módosításait ismerte meg a minimax algoritmusnak, és írja melléjük, hogy ezek milyen szempontból javítanak annak működésén?

Negamax, Alfa-béta.

Javítás?


(7.jatek.pdf)

5. A rezolúció fő részei

- globális munkaterület: aktuális klózalmaz
- kiindulási érték: az „axiómák \Rightarrow célállítás” klózái
- terminálási feltétel:
 - sikeres: üres klóz
 - sikertelen: nincs újabb rezolvens klóz
- kereső szabály: rezolvens képzés
- vezérlési stratégia: elég a nem-módosítható
- heurisztika: jó lenne a hatékonyság miatt, de sajnos nincs

(10.auto_kovetkeztet.pdf - 2. oldal)

I. állapotér repr. a misszionárius - kannibál problémára



Misszionárius - kannibál probléma

Elég csak a bal partot jegyezni, az átkelés nem külön állapot

n misszionárius és n kannibál át akar kelni egy folyón egy h személyes csónakban ...

Állapottér: $AT = \text{rec}(m : [0..n], k : [0..n], c : \mathbb{L})$ $I(m, k) \Rightarrow I(n-m, n-k)$

invariáns: nincs emberevés, azaz $I(m, k) \equiv m=k \vee m=0 \vee m=n$

Kezdőállapot: (n, n, igaz) Célállapot: $(0, 0, \text{hamis})$

Műveletek: $I(\text{this}.m, \text{this}.k) \wedge I(\text{this}.m-x, \text{this}.k-y) \Rightarrow I(x, y)$

<p>Oda$(x, y): AT \rightarrow AT$</p> <p>HA $\text{this}.c$ és $0 \leq x \leq \text{this}.m$</p> <p style="padding-left: 20px;">és $0 \leq y \leq \text{this}.k$ és $0 < x+y \leq h$</p> <p style="padding-left: 20px;">és $I(\text{this}.m-x, \text{this}.k-y)$</p> <p>AKKOR $\text{this}.c := \text{hamis} :$</p> <p style="padding-left: 40px;">$\text{this}.m := \text{this}.m - x :$</p> <p style="padding-left: 40px;">$\text{this}.k := \text{this}.k - y$</p>	<p>Vissza$(x, y): AT \rightarrow AT$ ($\text{this} : AT$)</p> <p>HA $\neg \text{this}.c$ és $0 \leq x \leq n - \text{this}.m$</p> <p style="padding-left: 20px;">és $0 \leq y \leq n - \text{this}.k$ és $0 < x+y \leq h$</p> <p style="padding-left: 20px;">és $I(\text{this}.m+x, \text{this}.k+y)$</p> <p>AKKOR $\text{this}.c := \text{igaz} :$</p> <p style="padding-left: 40px;">$\text{this}.m := \text{this}.m + x :$</p> <p style="padding-left: 40px;">$\text{this}.k := \text{this}.k + y$</p>
--	--

(2.allrepr.pdf - 5. oldal)

II. (2,2) átlagoló eljárás egy adott játékfára. Soron következő lépés.

2012. 05. 30.

3. Mikor nevezünk egy gráf kereső algoritmust A^c algoritmusnak, és melyek a legfőbb tulajdonságai

$f = g+h$ és $0 \leq h$ és

$h \leq h^*$ és

$h(n) - h(m) \leq c(n, m)$

- optimális megoldást ad, ha van
- ugyanazt a csúcsot nem terjeszti ki kétszer

(5.grafker.pdf - 3. oldal)

5. Hogyan néz ki egy általánosított perceptron, és hogyan számolja ki a kimeneti értéket

II. Visszalépés keresés lejtése egy gráfon, mennyi visszalépés volt, mi a megoldási út, szömozoni kellett bejárás szerinti sorrendbe, és x-el jelölni azokat a csúcsokat ahol volt visszalépés

2012. 05. 23.

I. Állapottér reprezentáció utazógynökre (n darab város) és becsülje meg az állapottérnek és problémának a méretét.

II. Hegyműszű algoritmus Hanoi torony rep. gráfjában(ami a diákon is van) adott heurisztika mellett $\sum_{i=0}^3 i \cdot v(i)$, ha két csucs között nem dönt heurisztika akkor a baloldali csúcsot részesítsük előnyben

//2015-----

állapotr reprezentációhoz a kockás volt de nem úgyanúgy mint a din hanem csak 3 móvelet volt és nem volt olyan h handempty

a 3. gyak feladat meg alpha-béta algoritmus

elméleti:

1. mondj példát arra amikor a problémátér és az állapottér megegyezik, meg arra amikor nem

2. mi az elsőrendű stratégia, csoportjai példák

Ha ez az elsődleges vezérlési stratégiákra vonatkozik, akkor: független a feladattól és annak modelljétől: nem merít sem a feladat ismereteiből, sem a modell sajátosságaiból.

- módosítható: visszalépéses keresések, gráfkeresések
- nem módosítható: lokális keresések, evolúciós algoritmus, rezolúció (3.lokaliter.pdf - 1. oldal)

3. felügyelt nem felügyelt tanulás. - ez gépi tanulás, majd kidolgozom később FÁ

4. mélységi gráfbejárás meg visszalépéses keresés mélységi számmal (vagy hogy mondják) összehasonlítás

What?

I. valami tic tac toe játékok állapot reprezentációja (4x4x4-es kocka részpontjaiba pakolgatunk x-et meg o-t. ha 5 egymás melletti vízszintes vagy függőleges vagy átlós kijön vagy a kocaka átlójában van 5 egyforma akkor nyert)

II. az átlagos algoritmus példa gráfon és az A algoritmus levezetése egy táblázatba kellett tölteni meg a hisztogrammot kellett kitölteni még

2015.06.01

1, Heurisztika, mire jó, KR-ben hol és mire használjuk?

A heurisztika olyan, a feladathoz kapcsolódó ötlet, amelyet közvetlenül építünk be egy algoritmusba azért, hogy annak eredményessége és hatékonysága javuljon, habár erre általában semmiféle garanciát sem ad.

Erős heurisztika nélkül nincs sok esély a cél megtalálására.

- Jó heurisztikára épített kiértékelő függvénnel elkerülhetőek a zsákutcák.

Hiányos

(3. lokalker.pdf - 4. oldal)

2, Redukció és dekompozíció különbségei és hasonlóságai?

TODO

3, Minimax módosításainak felsorolása. Min javítanak ezek?

Nem találom

4, Szélességi és mélységi korlátos mélységi bejárás összehasonlítása.

TODO

6, Visszafelé haladó szabályalapú logikai reprezentáció, formai megszorításai.

□ Tény:

- $L_1 \wedge \dots \wedge L_n$ alakú univerzálisan kötött kifejezés, ahol L_1, \dots, L_n literálok.

□ Szabályok:

- $W \rightarrow L$ alakú univerzálisan kötött kifejezések, ahol L egy literál, a W pedig ÉVF kifejezés

□ Cél:

- egzisztenciálisan kötött tetszőleges ÉVF kifejezés

ÉS/VAGY formájú (ÉVF) kifejezések:

- literálok
- $A \wedge B, A \vee B$ alakú formulák, ahol az A és B is ÉVF kifejezés.

(10.auto_kovetkeztet.pdf - 7. oldal)

I, állapottér reprezentáció: 3 kocka, 2 golyó, 1 robotkar, ami golyót mozgat dobozok között. Kezdetben X-ben A, Y-ban B golyó van, Z üres. Cél: X és Y tartalmának cseréje.

II, Adott gráfon visszalépéses keresés. (Sorszámozni az éppen aktuális csúcsot [egy csúcsnál több sorszám is lehet], X-szel jelölni azokat a csúcsokat, ahonnan visszalép, megadni a visszalépések számát és a végeredmény utat.

III, Adott ÉS/VAGY gráfból közönséges irányított gráfot készíteni.

2. Előre:

ÉS/VAGY formájú (ÉVF) kifejezések:

- literálok
- AB, AB alakú formulák, ahol az A és B is ÉVF kifejezés.

Tény: univerzálisan kötött tetszőleges ÉVF kifejezés

Szabályok: $L \rightarrow W$ alakú univerzálisan kötött kifejezések, ahol L egy literál, a W pedig ÉVF kifejezés

Cél: $L_1 \vee \dots \vee L_n$ alakú egzisztenciálisan kötött kifejezés, ahol L_1, \dots, L_n literálok.

3. Visszafele:

Tény: $L_1 \wedge \dots \wedge L_n$ alakú univerzálisan kötött kifejezés, ahol L_1, \dots, L_n literálok.

Szabályok: $W \rightarrow L$ alakú univerzálisan kötött kifejezések, ahol L egy literál, a W pedig ÉVF kifejezés

Cél: egzisztenciálisan kötött tetszőleges ÉVF kifejezés

ÉS/VAGY formájú (ÉVF) kifejezések:

- literálok
- AB , AB alakú formulák, ahol az A és B is ÉVF kifejezés.

#####

2015. jan. 15:

elmélet:

1. milyen modellezési megoldásokat vettünk a félév során

megoldások:

- állapotreprézntáció
- keresési rendszerek
- redukció
- dekompozíció
- modellezés mesterséges neurális hálókkal (perceptron, MLP és Hopfield)

2. szelekció és arra 3 példa hogy mire használtuk - vmi ilyesmi

Szelekció: Kiválasztunk néhány (lehetőleg rátermett) egyedet szülőnek.

+példák

(9.evolucio.pdf - 1. oldal)

3. vmi rezolúciós levezetés leírása

4. visszalépéses keresés leírása és heurisztikái

A visszalépéses keresés egy olyan KR

- globális munkaterülete:
 - egy út a startcsúcsból az aktuális csúcsba (ezen kívül az útról leágazó még ki nem próbált élek)
 - Kezdetben a startcsúcsot tartalmazó nulla hosszúságú út
 - terminálás célcsúccsal vagy startcsúcsból való visszalépéssel
- keresés szabályai:
 - a nyilvántartott út végéhez egy új (ki nem próbált) él hozzáfűzése, vagy a legutolsó él törlése (visszalépés szabálya)
- vezérlés stratégiája a visszalépés szabályát csak a legvégső esetben alkalmazza

Heurisztikák:

- sorrendi szabály: sorrendet ad az aktuális út végpontjából kivezető élek (utak) vizsgálatára

- vágó szabály: megjelöli azokat az aktuális út végpontjából kivezető éleket (utakat), amelyeket nem érdemes megvizsgálni
(4.visszalep.pdf - 1. oldal)

5. rezolúció másodlagos stratégiái

A rezolúciós stratégia a rezolúció alapalgorithmusát kiegészítő olyan előírás, amely

- korlátozza egy adott pillanatban előállítható rezolvensek körét, vagy
- szabályozza a rezolvens képzés sorrendjét.

A rezolúciós stratégiák a rezolúciónak, mint kereső rendszernek, a másodlagos vezérlési stratégiái: csak klóz alapú reprezentáció esetén értelmezhetőek.

A korlátozásnál megsérülhet a teljesség: azaz nem vezethető le minden esetben az üres klóz

(10.auto_kovetkeztet.pdf - 3. oldal)

6. és/vagy fát mire használtuk a félév során

A közönséges irányított gráfok útkereső algoritmusait szeretnénk adaptálni az ÉS/VAGY gráfbeli keresési problémákhoz: megoldás gráf kereséséhez.

A megoldás gráf egy startcsúcsból kiinduló hiper-gráf. A startcsúcsból kiinduló hiper-gráfokat megadhatjuk a bejárásaikkal, ennél fogva ábrázolhatóak közönséges irányított utakkal.

Ez lehetővé teszi, hogy minden ÉS/VAGY gráfnak megfeleltethessünk egy olyan közönséges -gráfot, amelynek megoldási útjai az ÉS/VAGY gráfbeli megoldás gráfok bejárásai, és ezeket közönséges útkereső algoritmusokkal megkereshetjük.

Nem tudom, hogy jó válasz-e

(6.2.dekomp.pdf - 4. oldal)

1) Sorolja fel a hegymászó algoritmus hátrányait, és írja le, milyen módon lehet ezeken javítani! (GT 3. dia)

hátrányok:

1. Csak erős heurisztika esetén lesz sikeres (zsákutcába kerülhet)
2. (azonos értékű szomszédos csúcsok között) található körön, végtelen működésbe eshet.

javítások:

1. zsákutca elkerüléséhez szimultán hűtés
2. deadlock kiszűrése tabu halmazzal

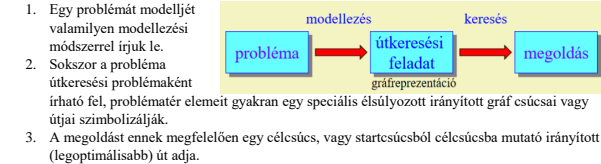
2. Mikor veszítheti el a gráfkeresés során a feszítőfa a korrektséget? Hogyan korrigálható ez, és mi melyik megoldást választottuk?

Abban az esetben veszíti el a fa a korrektséget, ha egy adott csúcsot már az algoritmus futása folyamán egyszer kiterjesztettünk, majd ezt követően újra kiterjesztesre kerül.

Csokkeno kiterjesztes hasz

3. Jellemezze és értékelje az A* algoritmus futási idejét! Ismer-e olyan gráfkereső algoritmusokat, amelyek jobb futási időt produkálnak? Jellemezze ezek futási idejét!
4. Mi az a hiper út, mit tekintünk a hiperút bejárásának és hogyan állítható elő?
5. Adja meg az evolúciós algoritmus! Röviden jellemezze az ebben található evolúciós operátorokat!
6. Hogyan lehet válaszadásra is alkalmazni a logikai következtetéseket. Mi volt ennek a módszere a rezolúciónál, és mi a szabály alapú következtetésnél?

Kapcsolat az útkeresési problémák, a gráfrepresentáció és a keresőalgoritmusok között



Milyen formában jelennek meg az általános keresőrendszer komponensei egy gráfkereső algoritmus esetén? • 1. keresőgráf (G) : a reprezentációs gráf eddig bejárt és eltárolt része • 2. nyílt csúcsok halmaza (NYLT) : kiterjesztésre várakozó csúcsok, amelyeknek gyerekeit még nem vagy nem eléggé jól ismerjük • 3. kiterjesztett csúcsok halmaza (ZART) : azok a csúcsok, amelyeknek a gyerekeit már előállítottuk • 4. kiterjesztés (T) : egy csúcs összes gyermekének előállítása a hozzájuk vezető élekkel együtt • 5. kiértékelő függvény (f: NYLT → R) : kiválasztja a megfelelő nyílt csúcsot kiterjesztésre

Problémátér vs. állapotér: A problémátér elemeit többnyire a start csúcsból kiinduló utak szimbolizálják. Algoritmusnak nincs állapottere, csak problémák, úgy pedig Hanoi-tornyai vs 8 királynő, mert a Hanoi-nál az, hogy a 2. rúdon van 1 karika, az egy állapot, de a problémátérbe meg az tartozna, hogy hogyan jutunk el oda. A 8 királynőnél meg a kettő ugyanaz. A5-ön van királynő, az egy állapot, meg a problémátér egy eleme is. Az állapotér-reprezentáció és a problémátér között szoros kapcsolat áll fenn, de az állapotér többnyire nem azonos a problémátérrel.

Keresőrendszer algoritmus. komponensei:

A globális munkatérületen a reprezentációs gráf egy részgráfja jelenik meg.

- vagy csak egy csúcs, esetleg annak közvetlen szomszédjai
- vagy a startcsúcsból kivezető egyik út kezdőszakasza
- vagy a startcsúcsból kivezető összes út kezdőszakasza

A keresési szabályok ezt a globális munkatérületen tárolt részgráfot módosítják

- lecserélnek egy csúcsot valamelyik szomszédjára

- hozzávesznek egy tárolt út végéhez egy onnan kiinduló élt

- törölnek egy tárolt út végétől egy élt

A vezérlési stratégia a fenti szabályokból válogat.

Feltételek terminálásra gráfkereső algonál: A GK véges delta-gráfban mindig terminál. Ha egy véges delta-gráfban létezik megoldás, akkor a GK megoldás megtalálásával terminál.

Milyen eredményeket mondunk ki az általános gráfkereső algoritmusról? • A GK gráfban a működés során egy csúcsot legfeljebb véges sokszor terjeszt ki. (tehát a körökre nem érzékeny) • A GK véges gráfban mindig terminál. • Ha egy véges gráfban létezik megoldás, akkor a GK megoldás megtalálásával terminál.

Mit tesz az általános gráfkereső algoritmus akkor, amikor egy már korábban felfedezett csúshoz talál minden addignál olcsóbb utat?

Ha $m \in G$ és $g(m) < g(n) + c(n, m)$, akkor:

$\pi(m) = n$ és $g(m) = g(n) + c(n, m)$, és az m csúcsot nyílttá tesszük. Ha a kiértékelő függvény csökkenő, a korrektség megától helyreáll.



!Globális munkatérület: Lokális keresés: egy csúcs és annak a szűk környezete | Visszalépéses keresés: egy út a startcsúcsból az aktuális csúcsba és az arról leágazó, még ki nem próbált élek | Gráfkeresés: egy olyan részgráf, a startcsúcsból induló, már feltárt út

!Keresési szabály: Lokális keresés: az aktuális csúcsot minden lépésben egy, az annak környezetében lévő, "jobb" csúccsal cseréljük le | Visszalépéses keresés: a nyilvántartott út végéhez egy új, még ki nem próbált élt hozzáfűzése, vagy a legutolsó élt törlése (visszalépés) | Gráfkeresés: az egyik útvegy csúcs kiterjesztése

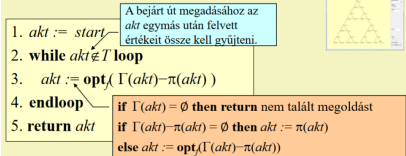
!Vezérlési stratégia: Lokális keresés: a "jobbság" eldöntésére kiértékelő függvényt használ, ami remélhetőleg annál jobb értéket ad egy csúcsra, minél közelebb van a célhoz Visszalépéses keresés: a visszalépés szabályát csak a legvégső esetben alkalmazza. | Visszalépés feltételei: zsákutca, zsákutca torkolat, kör, mélységi korlát | Gráfkeresés: Mindig a legkedvezőbb csúcs kiterjesztésére törekszik

Hegymászó algoritmus vs tabu keresés: A hegymászó algoritmus lokális optimum hely körül, illetve ekvidisztans felületen lévő körön végtelen működésbe eshet, ezt hivatott kiküszöbölni a tabu keresés a memória növelése által. A tabu keresés nyilvántart egy tabu halmazt és az optimális csúcsot.

Mindig az aktuális csúcs legjobb, de nem a tabu halmazban lévő gyerekére lép. Ha az aktuális csúcs jobb, mint a jelenlegi optimális csúcs, akkor kicseréljük az optimális csúcsot az aktuális csúcsra. Az algoritmus minden lépésben frissíti a sorszerkeztet tabu halmazt az aktuális csúccsal. Az algoritmus terminál, ha az optimális csúcs célsúcs, vagy ha az optimális csúcs már régóta nem változott.

Hegymászó algoritmus: Olyan lokális keresési algoritmus, ami a start csúcsból kiindulva mindig az aktuális csúcs legjobb gyerekére lép, ami lehetőleg nem a szülője.

□ Mindig az aktuális (akt) csúcs legjobb gyermekére lép, amelyik lehetőleg nem a szülője.

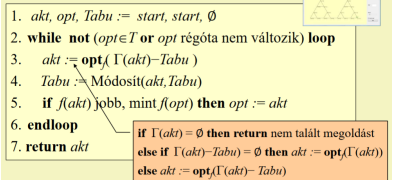


• Az eredeti hegymászó algoritmus nem zárja ki a szülőre való lépést, viszont nem engedi meg, hogy az aktuális csúcsot egy rosszabb értékű csúcsra cseréljük (ilyenkor a keresés inkább leáll).

Melyik problémáját küszöböli ki a tabu keresés a hegymászó módszernek + probléma jellemzése + tabu megoldásának jellemzése: A hegymászó algoritmus lokális optimum hely körül, vagy ekvidisztans felületen lévő körön végtelen működésbe eshet. Ezt küszöböli ki a tabu keresés.

Nyilvántartjuk az optimális csúcsot, és a tabu halmazt. Minden lépésben: • Az aktuális csúcs legjobb gyerekére lép, ami nincs a tabu halmazban • Ha az aktuális csúcs jobb, mint az optimális, akkor opt = akt • akt-et hozzáadjuk a tabu halmazhoz.

Tabu algoritmus:



Állapotér-reprezentáció: • Állapotér: a feladat fókuszában álló adat-együttes (objektum) lehetséges értékeinek (állapotainak) halmaza, egy állapot többnyire egy összetett szerkezetű érték, amelynek gyakran meg kell felelnie egy invariáns állításnak is. • Műveletek: állapottól állapotba vezetnek □ megadásukhoz: előfeltétel és hatás-leírás □ invariáns tulajdonságot tartó lekpezés

• Kezdő állapot(ok) vagy azokat leíró kezdőfeltétel • Céllápot(ok) vagy célfeltétel

Állapotér gráfrepresentációval: Állapot-gráf (az állapotér-reprezentáció reprezentációs gráfja):

- állapot – csúcs • művelet hatása – irányított él • művelet költsége – élköltség
- kezdő állapot – startcsúcs • céllápotok – célsúcsok

• műveletsorozat hatása – irányított út

ÉS/VAGY gráf: A közönséges irányított gráfok útkereső algoritmusait szeretnénk adaptálni az ÉS/VAGY gráfbeli keresési problémákhoz: megoldás gráf kereséséhez. A megoldás gráf egy startcsúcsból kiinduló hiper-gráf. A startcsúcsból kiinduló hiper-gráfokat megadhatjuk a bejárásaikkal, ennél fogva ábrázolhatóak közönséges irányított utakkal. Ez lehetővé teszi, hogy minden ÉS/VAGY gráfna megfelelőtessünk egy olyan közönséges gráfot, amelynek megoldási útjai az ÉS/VAGY gráfbeli megoldás gráfok bejárásai, és ezeket közönséges útkeresőalgoritmusokkal megkereshetjük.

Olyan $R=(N,A)$ gráf, ahol: • N: a csúcs egy részproblémát jelöl • s startcsúcs: $s \in N$ • t célsúcsok: $t \in T \subseteq N$ • A: élköteg, ami a dekomponálható probléma csúcsából dekomponálással kapott részproblémák csúcsaiba vezet. Élköteg élei közötti kapcsolatokat fajtái: □ "ES": a probléma megoldásához annak minden részproblémáját meg kell oldani □ "VAGY": választhatunk, hogy melyik élköteg mentén oldjuk meg a problémát

Dekompozíciós reprezentációja tartalma: PD reprezentációjának nevezzük a probléma részekre bontásának szemléltetésre épülő feladat reprezentációt. Ez, a probléma illetve a részproblémák leírása mellett, a közöttük kapcsolatot teremtő úgynevezett redukciós operátorok megadását is jelenti.

Probléma-dekompozíció reprezentáció szemléltetésére is gráfokat használunk, a megfogalmazott feladatokat egy ÉS/VAGY gráfon történő keresési problémaként értelmezhetők. A reprezentációhoz meg kell adnunk: • a feladat részproblémáinak általános leírását, • az eredeti problémát, • az egyszerű problémákat, amelyekről könnyen eldönthető, hogy megoldhatók-e vagy sem, • dekomponáló műveleteket: D: probléma -> probléma+ és D(p)=-<p1, ..., pn>

Milyen okok akadályozzák egy probléma dekompozíciós reprezentációjának megfogalmazását?

• Dekomponáló műveleteket nagyon nehéz megtalálni. • Nem minden feladat dekomponálható.

• Könnyen kaphatunk hamis dekomponáló műveleteket. • Az egyszerű probléma felismerése sem mindig egyértelmű. • A megoldás kiolvasása sem nyilvánvaló.

Hiperút: A $n^+ \rightarrow M$ hiper-út, ($n \in N, M \in N$) olyan véges részgráf, amiben: hiper-út egy olyan véges részgráf, amelyben • M csúcsaiból nem indul hiper-él, • többi csúcsból egyetlen hiper-él indul, • bármelyik csúcs elérhető az n csúcsból egy közönséges irányított úton, • Hiper-út hossza a benne lévő élek száma.

Mit értünk egy hiperút bejárásán? Egy irányított hiper-út bejárása is egy sorozat, de ez csúcsok sorozatának sorozata, és ez nem egyértelmű.

Osztályozza a vezérlési stratégiákat: • Elsődleges vezérlési stratégia: □ független a feladattól és annak modelljétől: nem merít sem a feladat ismeretéből, sem a modell sajátosságából. 1. nemmodosítható (lokális keresések pl. evolúciós algoritmus, rezolúció) 2. modosítható (visszalépéses keresések) • Másodlagos vezérlési stratégia: □ nem függ a feladat ismereteitől, de épít a feladat modelljének általános elemeire. pl. Lineáris input stratégia (rezolúciós) • Heurisztika: a feladattól származó, annak modelljében nem rögzített, de a megoldást segítő speciális ismeret. (pl. Manhattan-heurisztika (8-as játéknál))

Milyen modellezési megoldásokat vettünk? • állapot reprezentáció • keresési rendszerek • redukció • dekompozíció • modellezés mesterséges neurális hálókkal (perceptron, MLP és Hopfield)

Milyen hatással van a heurisztika általában a kereső rendszerek működésére? A heurisztika olyan, a feladathoz kapcsolódó ötlet, amit közvetlenül építünk be egy algoritmusba azért, hogy annak eredményessége és hatékonysága javuljon, bár erre semmiféle garanciát nem ad.

Heurisztika a gráfkereséseknél: Heurisztikus függvénynek nevezzük azt a $h:N \rightarrow R$ függvényt, amelyik egy csúcsnál megbecsüli a csúcsból a célba vezető („hátralévő”) optimális út költségét.

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

Heurisztikus függvények tulajdonságai: • Nem-negatív: $h(n) \geq 0 \forall n \in N$ • Megengedhető (admissible): $h(n) \leq h^*(n) \forall n \in N$ • Monoton megszorítás: $h(n) - h(m) \leq c(n, m) \forall (n, m) \in A$ (következetes)

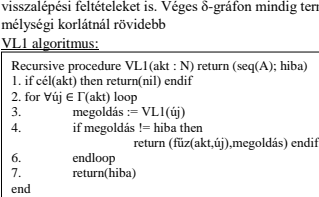
Milyen a jó heurisztika? • megengedhető: $h(n) \leq h^*(n)$ • jól informált: $h(n) \sim h^*(n)$ • monoton megszorítás: $h(n) - h(m) \leq c(n, m)$ • Változó heurisztikák: □ $f = g + \phi \cdot h$ B* algoritmus

Elnevezés	Definíció	Eredmények
Mélyégi	$f=g$, $c(n,m)=1$	Végtelen gráfokban mélyégi korláttal megoldást garantál
Szélességi	$f=g$, $c(n,m)=1$	Végtelen gráfokban a legrövidebb megoldást adja, egy csúcsot csak egyszer terjeszt ki.
Egyenletes	$f=g$	Végtelen gráfokban a legolcsóbb megoldást adja, egy csúcsot legfeljebb egyszer terjeszt ki.

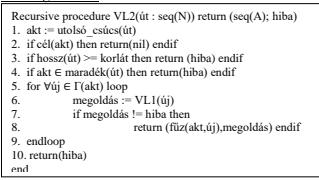
Melyek a visszalépéses keresés előnyei hátrányai: • Előnyök: □ Mindig terminál, talál megoldást, de csak a mélyégi korláton belül □ Könnyen implementálható □ Kicsi a memóriagénye • Hátrányok: □ Nem ad optimális megoldást (iterációba szervezhető) □ Kezdetben hozott rossz döntést csak sok visszalépés korrigál □ Egy zsákutca rész többször is bejárhat a keresés

Milyen eredményre képes a visszalépéses keresés első, illetve második változata? • VL1: Zsákutca és zsákutca torkolat visszalépési szabályokat implementáljuk. Véges körmentes gráfon mindig terminál és talál megoldást, ha van • VL2: A fenti kettőn kívül implementálja a kör és a mélyégi korlát visszalépési feltételeket is. Véges δ -gráfon mindig terminál, és talál megoldást, ha van olyan, ami a mélyégi korlátnál rövidebb

VL1 algoritmus:



VL2 algoritmus:



Elnevezés	Definíció	Eredmények
Előre tekintő gráfkeresés	f=h	Nincs említésre méltó tulajdonsága
A algoritmus	f=g+h, és h>=0	Megoldást ad, ha van, még végtelen gráfban is
A* algoritmus	f=g+h, és h <= h* és h <= h*	Optimális megoldást ad, ha van, még végtelen gráfokban is
A ^c algoritmus	f=g+h, és h>=0, és h<=h* és h(n)-h(m)<=c(n,m)	Optimális megoldást ad, ha van és ugyanazt a csúcsot nem terjeszti ki kétszer

Szimulált hűtés algoritmusa: A következő csúsz 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.

1. akt := start; k:= 1; i := 1
2. while not(akt ∈ T or f(akt) négóta nem változik) loop
3. if i > L _k then k := k+1; i := 1
4. új := select (f(akt)-n(akt))
5. if f(új)<=f(akt) or f(új)>f(akt) and e'((f(akt)-f(új))/Tk) > rand[0,1]
6. then akt := új
7. i:=i+1
8. endloop
9. return akt

Mit jelent kétszemélyes játékoknál a nyerő stratégia és milyen állítást mondtunk ki ezzel kapcsolatban?
• A játékokat állapotér-reprezentációval szokás leírni, és az állapot-gráfot faként ábrázolják
• Egy játékos nyerő stratégiája olyan elv, amit betartva az ellenfél minden lépésére tud olyan választ adni, hogy megnyerje a játékot.
• Nem-vesztő stratégia: A nyerő stratégia párja abban az esetben, ha a játék megengedi a döntetlent.
• A nyerő/nem-vesztő stratégia egy megfelelő ES/VAGY fa részgráfiával ábrázolható. Olyan részfa, ami egy olyan hiperút, amit a kezdő állás csúcsából vezet a nyerő/nem-vesztő végállások csúcsaiba
• Tehát: keresése egy ES/VAGY gráfbeli hiperút keresési probléma.
TÉTEL: Ha a döntlen nem megengedett, akkor a kétesélyes kétszemélyes játékokban az egyik játékos számára biztosan létezik nyerő stratégia. A kétszemélyes háromsélyes (-ahol megengedett a döntetlen) játékokban a nem vesztő stratégiát lehet biztosan garantálni.

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 min_i (NYILT) <F
 then n := min_s (mENYILT | f(m) < F)
 else n := min_i (NYILT); F := f(n)

endif elágazással.

A*,A^c, B algoritmus futási ideje: Egygráfkeresésmemória 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 időlegrosszabb esetben exponenciálisan függ a kiterjesztettcsúcsok 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, hogyegy A^c algoritmus ugyanazon a gráfon összességében kevesebb kiterjesztést végez, mint egy csúcsot többször is kiterjesztő A* algoritmus. A Balgoritmus futási ideje négyzetes, és ha olyan heurisztikus függvényt használ, mint az A*algoritmus(azaz megengedhető), akkor ugyanúgy optimális megoldást talál (ha van megoldás) és a kiterjesztett csúcsok száma (mellesleg a halmaza is)megegyezik az A*algoritmus által kiterjesztett csúcsokéval.

Mi történik egy evolúciós algoritmus egy iterációja során?
(milyen lépések hajtódnak végre)

- Szelekció: Kiválasztunk néhány (lehetőleg rátermett) egyedet szülőnek: selection(Entity[] from)
- Rekombináció: A szülőkből utódok készülnek úgy, hogy a szülők tulajdonságait öröklőjk a gyerekek: recombination(Entity[] entityGroup)
• Mutáció: Az utódok tulajdonságait kismértékben módosítjuk: mutation(Entity[] entityGroup)
• Visszahelyezés: Új populációt alakítunk ki az utódokból és a régi populációból: insertion(Entity[] to, Entity[] from)
• Inicializálás: Kialakítjuk a kezdeti populációt
• Terminálás: Addig futtatjuk az algoritmust, amíg el nem érjük a kívánt állapotot

Evolúció algoritmusa:

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

Hogyan történik az evolúciós algoritmusokban a visszahelyezés? A populáció az utódokkal történő frissítése: kiválasztja a populáció lecsereendő egyedeit, és azok helyére az utódokat teszi. □ utódképzési ráta(u) = utódok száma/populáció száma □visszahelyezési ráta(v) = lecsereendő egyedek száma/populáció száma
• Ha u=v: feltétlenül csere
• Ha u<v: egy utód több példányban is bekerülhet
• Ha u>v: az utódok közül szelektál
Adjon példát legalább három rekombinációs operátorra!
• Egyponτος keresztezés
• Többponτος keresztezés
• Egyenletes keresztezés
• Parciálisan illesztett keresztezés
• Ciklikus keresztezés
Populáció: A problémára adható néhány lehetséges választ, azaz a problémátér több egyedét tároljuk egyszerre. Ez a populáció. Kezdetben egy többnyire véletlen populációt választunk. A cél egy bizonyos összeg vagy egy jó populáció előállítása. Az egyedeket egy rátermettségi függvény alapján hasonlítjuk össze. A populációt lépésről lépésre javítjuk úgy, hogy a kevésbé rátermett egyedek egy részét a rátermettebekhez hasonló egyedekre cseréljük le. Ez a változtatás visszavonhatatlan. Ez tehát egy nem-módosítható stratégiájú keresés.

- **Felügyelt és felügyelet nélküli tanulás def + példák**
Egy neuron esetében a Δw kiszámítása az éppen vizsgált minta bemeneti értékeire és a neuron által kiszámított kimenetre támaszkodik.
 - *Felügyelt tanulás* esetén felhasználjuk a neuronnal a vizsgált minta alapján elvárt kimeneti értékét is
 - *Felügyelet nélküli tanulás* esetén a várt kimenetre nincs szükség

Példák:

Felügyelt tanulás (Delta szabály)	Felügyelet nélküli tanulás (Hebb szabály)
$\Delta w_{ij} = \eta \cdot o_i \cdot (t_j - o_j)$	$\Delta w_{ij} = \eta \cdot o_i \cdot o_j$

Jejmagyarázat:

η: tanulású együttható, a tanulás sebességét befolyásolja.
o: az i-edik neuron számított kimenete, és egyben a j-edik neuron i-edik bemenete is. || w_{ij}: a j-edik neuron bemenetének súlya
o_j: a j-edik neuron számított kimenete || t_j: a j-edik neuron várt kimenete

<p>if <i>h(n)<min_{m∈Γ(n)} (c(n,m)+h(m))</i></p> <p>then <i>h(n) := min_{m∈Γ(n)} (c(n,m)+h(m))</i></p> <p>else <i>for</i> $\forall m \in \Gamma(n)$-re loop</p> <p style="padding-left: 40px;"><i>if</i> <i>h(n)- h(m)>c(n,m)</i> then <i>h(m) := h(n)- c(n,m)</i></p> <p>endloop</p>	<ul style="list-style-type: none">• A h megengedhető marad • A h nem csökken • A monotonon megszorításos élek száma nő <p>B’ ALGORITMUS</p>
--	--

<p>Mohó A algoritmus</p> <ul style="list-style-type: none">□ Nincs mindig szükség az optimális megoldásra. <ul style="list-style-type: none">– Ilyenkor a mohó <i>A’</i> algoritmus is használható, amely rögtön megáll, ha célsúcs jelenik meg a <i>NYILT</i>-ban. □ A mohó <i>A’</i> algoritmus csak a megoldás megtalálását garantálja. De belátható <ul style="list-style-type: none">– Ha h megengedhető és $\forall t \in T: \forall (n,t) \in A: h(n)+\alpha \leq c(n,t)$, akkor a talált megoldás költsége: $g(t) \leq h'(s)+\alpha$. □ A mohó <i>A’ algoritmus</i> megengedhető heurisztika mellett akkor garantálja az optimális megoldást is, <ul style="list-style-type: none">– ha $\forall t \in T: \forall (n,t) \in A: h(n) \leq c(n,t)$ vagy – ha <i>h</i> monoton és $\exists \alpha \geq 0: \forall t \in T: \forall (n,t) \in A: h(n)+\alpha \leq c(n,t)$	<p>Minimax algoritmus: 1. A játékának az adott állás csúcsából leágazó részfáját felépítjük néhány szintig. 2. A részfa leveleit kiértékeljük aszerint, hogy azok számunkra kedvező, vagy kedvezőtlen állások. 3. Értékeket felfuttatjuk a fában: • A saját (MAX) szintek csúcsaihoz azok gyermekeinek maximumát: szülő := max (gyereki <i>.....</i> gyerek <i>k</i>) • Az ellenfél (MIN) csúcsaihoz azok gyermekeinek minimumát: szülő := min (gyereki <i>.....</i> gyerek <i>k</i>) 4. Soron következő lépésünk ahhoz az álláshoz vezet, ahonnan a gyökérhez felkerült a legnagyobb érték.</p>
---	---

Átlagoló kiértékelés: Célja a kiértékelő függvény esetleges tévedéseinek simítása.

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 részfat, majd ettől a szinttől kezdve egy adott szintig (maximális mélység) csak azon csúcsokat terjesztjük ki, amelyekre nem teljesül a nyugalmi teszt: |f(szülő) f(csúcs)|< K. Szelektív kiértékelés: Célja a memória-igény csökkentése. Elkülönytjük a lényeges és lényegtelen lépéseket, és csak a lényeges lépéseknek megfelelő részfat építjük fel. Ez a szétválasztás heurisztikus ismeretekre épül.
Negamax algoritmus: Negamax eljárást könnyebb implementálni. – Kezdetben (–1)-gyel szorozzuk azon levélcúcsok 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*)

Alfa-béta algoritmus

- Visszalépéses algoritmus segítségével járjuk be a részfat (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 *α* := –∞, és *β* := +∞.
- **Visszalépéskor** az éppen elhagyott (gyermek) csúcs értéke (felhozott érték) módosíthatja a szülő csúcs értékét:
 - a MAX szintjein: *α* := *max(felhozott érték, α)*
 - a MIN szintjein: *β* := *min(felhozott érték, β)*
- **Vágás:** ha az úton van olyan *α* és *β*, hogy *α*≥*β*.

Sorolja fel, milyen módosításait ismerte meg a **minimax** algoritmusnak, és írja melléjük, hogy ezek melyik szempontból javítanak annak működésén?
• Átlagoló kiértékelés: A kiértékelő függvény esetleges tévedéseit simítja ki
• Váltakozó mélységű kiértékelés: A kiértékelő függvény minden ágon reális értéket mutat
• Szelektív kiértékelés: A memóriaigényt csökkenti (csak a lényeges lépéseket értékelik) □ Negamax: könnyebb implementálni □ Alfa-béta: kisebb a memóriaigény, és a futási idő is

Gépi tanulás

Egy algoritmus akkor tanul, ha egy feladatcsoport minta (tanító) feladataira végrehajtott futtatásai során mind a mintafeladatokra, mind a feladatcsoport más feladataira adott megoldása működésről működésre egyre jobb lesz. A tanulás célja az, hogy egy $\varphi: X \rightarrow Y$ leképezést tanuljunk meg azáltal, hogy a helyettesítéséhez egy olyan $f: P \times X \rightarrow Y$ leképezést (P a paraméterek halmazát jelöli) konstruálunk, amelyek minél kisebb hibával közelíti a φ -t. (Ezekről eltérő, egyedi tanuló algoritmusokkal is találkozhatunk, mint például Mérő B' gráffereső algoritmus, amelyek a kezdetben megadott neurisztikát módosítják.) A tanulóshoz az X halmazból vett tanító mintákat használunk.

Felügyelt tanulás

Felügyelt tanulás esetén a tanító minták φ szerinti eredményeit is ismerjük: azaz a tanító halmaz ilyenkor a $T = \{(x_n, y_n), n=1,...,N$ és $y_n=\varphi(x_n)\}$. A φ és az f leképezéseknek a tanító mintákra vett hibáját az $L(\theta, T) = \sum_{n=1}^N l(f(\theta, x_n), y_n)$ hibafüggvény mutatja, ahol $l: Y \times Y \rightarrow \mathbb{R}$ egy alkalmas távolság függvény. A tanulási folyamat célja egy olyan θ -nak a megtalálása, amelyre az $L(\theta, T)$ kellően kicsi. Attól függően, hogy a hibafüggvény hiper-paramétereit ($f(\theta, x)$, $l(x, y)$, stb.) hogyan választjuk meg, különféle tanulási módszerekhez jutunk.

- A *K legközelebbi szomszédok* (KNN) módszernél az $f(\theta, x)$ értéke úgy áll elő, hogy vesszük tanító minták közül azt a K darabot, amelyek x_n -jei a legközelebb esnek x -hez, és ezen K darab minta y_n -jeinek számtani közepét képezzük. Az $f(\theta, x)$ kiszámítása ennél a módszernél lassú. A θ -t itt tulajdonképpen a minták jelentik. A módszer K -ra érzékeny.
- A *véletlen erdő* (RF) módszer először K darab döntési fát épít fel a tanító minták alapján úgy, hogy egy fa építéséhez a tanító mintáknak is, és a minták attribútumainak is egy-egy véletlen részhalmazát használja csak fel.¹ Egy véletlen erdő birtokában egy x bemenetre megállapíthatjuk, hogy az x az egyes döntési fák melyik levelére képződik le. Vesszük az ezen levelekhez tartozó tanító példák részhalmazait (T_k), és az $f(\theta, x)$ értékét az y_n -ek súlyozott összegeként számoljuk úgy, hogy a súly azon $1/|T_k|$ értékek számtani közepe, amelyre a T_k tartalmazza az y_n -hez tartozó x_n -t. Az $f(\theta, x)$ kiszámítása (a véletlen erdő ismeretében) gyors. A θ -t itt a véletlen erdő adja.
- A *mély neurális hálózatok* (DNN) esetében az $f(\theta, x)$ egy $g_K(\theta_K, g_{K-1}(...g_2(\theta_2, g_1(\theta_1, x_n))...))$ alakú összetett függvény, amellyel rétegről rétegre történik a leképezés: $x_n^{(k)} = g_k(\theta_k, x_n^{(k-1)})$. Egy tipikus (sűrűn rétegetl) változata ennek az, amikor a θ_k paraméterek ún. W_k súlymátrixok, és egy réteg az $x_n^{(k)} = h_k(W_k \cdot x_n^{(k-1)} + b_k)$ képlettel számolható, ahol h_k egy folytonos differenciálható függvény, b_k az m dimenziós ún. bias vektor. Az $f(\theta, x)$ kiszámítása ennél a módszernél gyors, a súlyok betanulása gradiens módszer alapján történik.

¹ Egy döntési fa egy $X \rightarrow Y$ leképezést ír le, ahol egy X típusú bemenet attribútumok sorozatával adható meg. A fa csúcsai egy-egy attribútummal vannak címkézve, és annyi gyereke van, ahány értéket az adott attribútum felvehet (illetve ahány tartományát megkülönböztetjük az értékeknek). A fa levelei egy-egy kimeneti értéket képviselnek. Egy bemeneti elem a fa egy levelére képződik le azáltal, hogy a fa gyökerétől elindulva úgy haladunk lefelé, hogy minden csúcsnál megvizsgáljuk, hogy a bemenetnek mi az adott csúcs attribútumára adott értéke, és ezen értéknek megfelelő gyerekcsúcsra lépünk. Amikor levélcsúcshoz érünk, az megadja a keresett kimeneti értéket.

A döntési fát megadott tanító mintából és azok ismert attribútumaiból építjük fel úgy, hogy a fa csúcsai a tanító példákat hierarchikusan osztályozzák. Minden csúcschoz a tanító példák egy részhalmaza tartozik, amelyeket a csúcs attribútuma alapján osztályozunk, és ezen osztályokat a csúcs gyerekcsúcsaihoz rendeljük. Egy levélcsúcs kimeneti értéke az, amellyel a levélcsúcschoz került példák többsége rendelkezik.

Nem felügyelt tanulás

Nem felügyelt tanulás esetében a tanító mintákhoz nem tartoznak kimeneti értékek, ezek nélkül próbálunk valamilyen általánosítható ismerethez jutni a $T = \{x_n, n=1,...,N\}$ tanító mintákat nyújtó X halmaz elemeiről.

A *klaszterezés* során részekre (klaszterekre) bontjuk az X halmazt úgy, hogy egy klaszteren belül minél hasonlóbb elemeket találjunk, két különböző klaszterből való elemek pedig minél kevésbé legyenek hasonlóak.

- A *k-means algoritmus* egy hard klaszterező módszer, azaz minden elem pontosan egy klaszterhez tartozhat. Előre meg kell adni a klaszterek számát. A klaszterek középpontja a klaszterhez tartozó minták átlaga. A cél, hogy a klaszterbe tartozó mintáknak a klaszter középpontjától vett távolságnégyzeteinek összege minimális legyen. Ezt a célt iterációval érjük el: minden lépésben meghatározzuk a klaszterek középpontjaihoz tartozó (azokhoz legközelebb eső) minták halmazait, majd ezen halmazok középpontjaiba igazítjuk a klaszter középpontokat. Mivel így csak lokális optimumokat tudunk előállítani érdemes véletlen kezdeti beállításokkal többször futtatni.
- A *téma modellezés* egy soft klaszterező eljárás, azaz a mintákról azt mondja meg, hogy milyen mértékben tartoznak egy klaszterhez. Itt a minták dokumentumok, a klaszterek pedig a témák. Az LSA (Latent Semantic Analysis) egy szó-dokumentum mátrix szinguláris érték felbontásaként áll elő. A bemeneti mátrixban minden sor egy szóhoz, minden oszlop egy dokumentumhoz tartozik, és a mátrix értékei, hogy az adott szó hányszor fordult elő az adott dokumentumban. A szinguláris érték felbontás eredményeként megkapjuk a szó-téma és a téma-dokumentum mátrixokat, amik hasonlóan épülnek fel.

A *dimenzió csökkentés* során adatpontjainkat egy alacsonyabb dimenziós térbe képezzük le. Ennek célja többek között lehet az adatok láttatása, vagy a lényeges információ kiemelése (a zaj csökkentése). Sokszor előfordul, hogy kisebb dimenziós adatok egy nagy dimenziós térben vannak.

- A *főkomponens analízis* az adatpontokat egy olyan koordinátarendszerbe transzformálja, ahol az első tengely mentén megtartjuk a lehető legtöbb szórást az adatokból, a második tengelyen a második legtöbb szórást, és így tovább. A tengelyeket főkomponenseknek hívjuk. Mivel a főkomponensek a megtartott szórás szerint rendezettek, az első k főkomponens megtartásával egy olyan dimenzióscsökkentéshez jutunk, ami a lehető legtöbb szórást tartja meg az adatokból. Ha az adatpontokat egy mátrix soraiba tesszük, és nulla átlagúra hozzuk, akkor a főkomponensek e mátrix kovariancia mátrixának sajátvektorai, amik szinguláris érték felbontással az input mátrixból könnyen kiszámíthatók.

Az *autoenkóderek* olyan neuronhálók, melyek célja egy, az input dimenziójánál rendszerint kisebb dimenziós reprezentáció (*kód*) tanulása az input rekonstrukálásával mint célfüggvénnyel. A legegyszerűbb, egy rejtett rétegű autoenkóderek a nemlinearitások ellenére egy főkomponens analízist valósítanak meg. Az autoenkóderek előnye, hogy tetszőlegesen bővíthetők, építhetőek. Néhány autoenkóder változat a zajtalanító autoenkóder, ami egy zajos inputból próbálja meg visszaállítani az eredeti zajtalan inputot, illetve a ritka autoenkóder, ahol a kódban egyszerre csak néhány neuron lehet aktív. A variációs autoenkóderek már egy valószínűségi modellt feltételeznek, és adott outputok helyett outputok eloszlását tanulják, így az autoenkódereket általánosíthatják.