1 AZ MI FOGALMA

1.1 Miről ismerhető fel az MI?

Megoldandó feladat: nehéz
- A feladat problématere hatalmas
-Szisztematikus keresés helyett intuícióra, kreativitásra (azaz heurisztikára) van szükségünk ahhoz, hogy elkerljük a kombinatorikus robbanást.
Szoftver viselkedése: intelligens (tárol ismeretet, automatikusan következtet, tanul, gépi látás, gépi cselekvés)
-Turing teszt vs. kínai szoba elmélet
-"mesterjelölt szintű" mesterséges intelligencia
Felhasznált eszközök: sajátosak
-Átgondolt reprezentáció a feladat modellezéséhez
-Heurisztikával megerősített hatékony algoritmusok
-Gépi tanulás módszerei

2 MODELLEZÉS ÉS KERESÉS

feladat → útkeresési probléma → megoldás
Feladat és útkeresési probléma között helyezkedik el a modellezés. Itt található az állapottér-reprezentáció, a probléma dekompozíció, a korlátprogramozási modell, és a logikai reprezentáció.
Az útkeresési problémát egy gráffal reprezentálhatjuk. Az útkeresési probléma és a megoldás között található a keresés. A keresésbe tartoznak a lokális keresések, a visszalépéses keresések, a gráfkeresések, az evolúciós algoritmus, a rezolúció és a szabályalapú következtetés.

2.1 Mire kell a modellezésnek fókuszálni

Problématér elemei: probléma lehetséges válaszai.
Cél: egy helyes válasz (megoldás) megtalálása
Keresést segítő ötletek (heurisztikák):
-Problématér hasznos elemeinek elválasztása a haszontalanoktól.
-Az elemek szomszédsági kapcsolatainak kijelölése, hogy a probléma tér elemeinek szisztematikus bejárását segítsük.
-Adott pillanatban elérhető elemek rangsorolása.
-Kiinduló elem kijelölése.

2.2 Útkeresési probléma

Egy útkeresési problémában a problématér elemeit egy olyan élsúlyozott irányított gráf csúcsai vagy útjai szimbolizálják, amelyik gráf nem feltétlenül véges, de a

csúcsainak kifoka véges, és van egy közös pozitív alsó korlátja az élek súlyának (költségének) (δ-gráf).
A megoldást ennek megfelelően vagy egy célcsúcs, vagy egy startcsúcsból célcsúcsba vezető út (esetleg a legolcsóbb ilyen) megtalálása szolgáltatja.
Számos olyan modellező módszert ismerünk, amely a kitűzött feladatot útkeresési problémává fogalmazza át.

2.3 Gráf fogalmak

-csúcsok, irányított élek:$\rightarrow$ N, A $\subseteq$ NxN
-él n-ből m-be $\rightarrow$ (n,m) $\in$ A (n,m $\in$ N)
-n utódai $\rightarrow$ γ (n) = { m $\in$ N | (n,m) $\in$ A }
-n szülei $\rightarrow$ π(n) $\in$ Π(n) = { m $\in$ N | (m,n) $\in$ A }
-irányított gráf $\rightarrow$ R=(N,A)
-véges sok kivezető él $\rightarrow$ | γ (n) | $< \infty$ ($\forall$ n $\in$ N)
-élköltség $\rightarrow$ c:A $\rightarrow$ $\mathbb{R}$
-δ tulajdonság ($\delta \in \mathbb{R}^+$) $\rightarrow$ c(n,m) $\geq \delta > 0$ ($\forall$(n,m)$\in$A)
-δ-gráf $\rightarrow$ δ-tulajdonságú, véges sok kivezető élű, élsúlyozott irányított gráf
-irányított út $\rightarrow$ $\alpha = (n,n_1),(n_1,n_2),...,(n_{k-1},m) = <n,n_1,n_2,...,n_{k-1},m>$ $n \rightarrow^{\alpha} m$, $n \rightarrow m$, $n \rightarrow M$ ($M \subseteq N$) $n \rightarrow m$, $n \rightarrow M$
-út hossza $\rightarrow$ az út éleinek száma: $|\alpha|$
-út költsége $\rightarrow$ $c(\alpha)=c^{\alpha}(n,m) := \sigma_{j=1..k} c(n_{j-1},n_j)$ ha $\alpha = <n=n_0,n_1,n_2,...,n_{k-1},m=n_k>$
-opt. költség $\rightarrow$ $c^*(n,m):=\min_{\alpha \in \{ n \rightarrow m \}} c^{\alpha}(n,m)$ {δ gráfokban ez végtelen sok út esetén is értelmes. Értéke ∞, ha nincs egy út se.}
-opt. költség2 $\rightarrow$ $c^*(n,M):=\min_{\alpha \in \{ n \rightarrow m \}} c^{\alpha}(n,M)$
-opt. költségű út $n \rightarrow^* m := min_c\{\alpha |\ \alpha \in \{n \rightarrow m\}\}$
-opt. költségű út $n \rightarrow^* M := min_c\{\alpha |\ \alpha \in \{n \rightarrow M\}\}$

2.4 Gráfreprezentáció fogalma

Minden útkeresési probléma rendelkezik egy (a probléma modellezéséből származó) gráfreprezentációval, ami egy (R,s,T) hármas, amelyben:
-R=(N,A,c) δ-gráf az ún. reprezentációs gráf,
-az s$\in$N startcsúcs,
-a T$\subseteq$N halmazbeli célcsúcsok.
És a probléma megoldása:
-egy t$\in$T cél megtalálása, vagy
-egy s$\rightarrow$T, esetleg s$\rightarrow^*$T optimális út megtalálása
(s-ből Tegyik csúcsába vezető irányított út, vagy s-ből T egyik csúcsába vezető legolcsóbb irányított út)
Az útkeresési problémák megoldásához azok a reprezentációs gráfjainak nagy mérete miatt speciális (nem determinisztikus, heurisztikus) útkereső algoritmusokra van szükség, amelyek:
-a startcsúcsból indulnak, amely az első aktuális csúcs;
-minden lépésben nem-determinisztikus módon új aktuális csúcso(ka)t választanak

a korábbi aktuális csúcs(ok) alapján (gyakran azok gyerekei közül);
-tárolják a már feltárt reprezentációs gráf egy részét;
-megállnak, ha célcsúcsot találnak vagy nyilvánvalóvá válik, hogy erre semmi esélyük.

2.5 Kereső rendszer (KR)

Procedure KR
1. ADAT:= kezdeti érték
2. while !terminálási feltétel(ADAT) loop
3. SELECT SZ FROM alkalmazható szabályok
4. ADAT := SZ(ADAT)
5. endloop
end
AHOL:
-ADAT: globális munkaterület, tárolja a keresés során megszerzett és megőrzött ismeretet
-alkalmazható szabályok: keresési szabályok, megváltoztatják a globális munkaterület tartalmát
-SELECT: vezérlési stratégia, alkalmazható szabályok közül kiválaszt egy "megfelelőt"

2.6 Kereső rendszerek vizsgálata

-helyes-e (azaz korrekt választ ad-e)
-teljes-e (minden esetben választ ad-e)
-optimális-e(optimális megoldást ad-e)
-idő bonyolultság
-tár bonyolultság

3 GÉPI TANULÁS

-Egy programozási feladat megoldásához meg kell adnunk a feladat modelljét és készítenünk kell egy ehhez illeszkedő algoritmust, amely a feladat megoldását előállítja.
-Gépi tanulással a modell (reprezentáció és/vagy heurisztika), illetve a megoldó algoritmus (többnyire annak bizonyos paraméterei) állhatnak elő automatikusan.
-A tanuláshoz a megoldandó probléma néhány konkrét esetére, a tanító példákra van szükség.
-A gépi tanulási módszereket három csoportba szokás sorolni: felügyelt-, nemfelügyelt, és megerősítéses tanulásra attól függően, hogy a tanító példák input-output párok, csak inputok, vagy input-hasznosság párok.

4 Állapottér-reprezentáció

Állapottér: a probléma leírásához szükséges adatok által felvett érték-együttesek (azaz állapotok) halmaza
-az állapot többnyire egy összetett szerkezetű érték
-gyakran egy bővebb alaphalmazzal és egy azon értelmezett invariáns állítással definiáljuk
Múveletek: állapotból állapotba vezetnek.
-megadásukhoz: előfeltétel és hatás leírása
-invariáns tulajdonságot tartó leképezés
Kezdőállapot(ok) vagy azokat leíró kezdeti feltétel
Végállapot(ok) vagy célfeltétel

4.1 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→AT honnan, hova ∈ {1,2,3}
HA a honnan és hova létezik és nem azonos, és van korong a honnan rúdon, és a hova rúd üres vagy a mozgatandó korong (honnan rúd felső korongja) kisebb, mint a hova rúd felső korongja, AKKOR this[honnan legfelső korongja] := hova

4.2 Állapottér-reprezentáció gráf-reprezentációja

δ-gráf állapot-gráf
-csúcs: állapot
-irányított él: művelet hatása
-élköltség: művelet költsége
startcsúcs kezdőállapot
célcsúcsok végállapotok
irányított út egy műveletsorozat hatása

4.3 Állapottér vs. problématér

-Az állapottér-reprezentáció és a problématér között szoros kapcsolat áll fenn, de az állapottér többnyire nem azonos a problématérrel.
-A problématér elemeit többnyire nem az állapotok, hanem a startcsúcsból induló különböző hosszúságú irányított utak.
-A hanoi tornyai problémánál például egy megoldást egy irányított út szimbolizál, amelyik a startcsúcsból a célcsúcsba vezet.
-Van amikor a megoldás egyetlen álapot (azaz csúcs), de ebben az esetben is kell találni egy odavezető operátor-sorozatot (azaz irányított utat).

4.4 Állapot-gráf bonyolultsága

Állapot-gráf bonyolultsága → Problématér mérete → Keresés számításigénye

A bonyolultság elsősorban a start csúcsból kivezető utak száma az oda-vissza lépések nélkül, amely nyilván függvénye a
-csúcsok és élek számának
-csúcsok ki-fokának
-körök gyakoriságának, és hosszuk sokféleségének

Ugyanannak a feladatnak több modellje lehet: érdemes olyat keresni, amely kisebb problémateret jelöl ki.
-Az előző reprezentációnál a problématér mérete, azaz a lehetséges utak száma, óriási. Készítsünk jobb modellt!
-Bővítsük az állapotteret, és használjunk új műveletet!
-Műveletek előfeltételének szigorításával csökken az állapot-gráf átlagos ki-foka.

4.5 Művelet végrehajtásának hatékonysága

A művelet kiszámítási bonyolultsága csökkenthető, ha az állapotokat extra információval egészítjük ki, vagy az invariáns szigorításával szűkítjük az állapotteret.

4.6 Hogyan "látja" egy keresés a reprezentációs gráfot?

Egy keresés fokozatosan fedezi fel a reprezentációs gráfot: bizonyos részeihez soha nem jut el, de a felfedezett részt sem feltétlenül tárolja el teljesen, sőt, sokszor torzultan "látja" azt: ha például egy csúcshoz érve nem vizsgálja meg, hogy ezt korábban már felfedezte-e, hanem új csúcsként regisztrálja, akkor az eredeti gráf helyett egy fát fog tárolni.

4.7 Reprezentációs gráf "fává egyenesítése"

Ha a keresés nem vizsgálja meg egy csúcsról, hogy korábban már felfedezte-e, akkor az eredeti reprezentációs gráf helyett annak fává kiegyenesített változatában keres.
Előny: eltűnnek a körök, de a megoldási utak megmaradnak.
Hátrány: duplikátumok jelennek meg, sőt a körök kiegyenesítése végtelen hosszú utakat eredményez.
A kétirányú (oda-vissza) élek szörnyen megnövelik a kiegyenesítéssel kapott fa méretét. De bármelyik keresésnél eltárolhatjuk egy csúcsnak azt a szülőcsúcsát, amelyik felől a csúcsot elértük. Így egy csúcsból a szülőjébe visszavezető él könnyen felismerhető és figyelmen kívül hagyható.

5 Probléma dekompozíció

Egy probléma dekomponálása során a problémát részproblémákra bontjuk, majd azokat tovább részletezzük, amíg nyilvánvalóan megoldható problémákat nem kapunk.
Sokszor egy probléma megoldását akár többféleképpen is fel lehet bontani részproblémák megoldásaira.

5.1 Dekompozíciós reprezentáció fogalma

A reprezentációhoz meg kell adnunk:
-a feladat részproblémáinak általános leírását
-a kiinduló problémát
-az egyszerű problémákat, amelyekről könnyen eldönthető, hogy megoldhatók-e vagy sem
-a dekomponáló műveleteket:
D: probléma $\rightarrow$ probléma$^+$
D(p)= $<p_1,...,p_n>$

5.2 A dekompozíció modellezése ÉS/VAGY gráffal

Egy dekompozíciót egy ún. ÉS/VAGY gráffal szemléltetjük: -egy csúcs egy részproblémát jelöl, a startcsúcs a kiinduló problémát, a célcsúcsok a megoldható egyszerű problémákat. -egy élköteg egy dekomponáló művelet hatását írja le, és a dekomponált probléma csúcsából a dekomponálással előállított részproblémák csúcsaiba vezet.

- egy élköteg élei mutatják meg, hogy a dekomponált probléma megoldásához mely részproblémákat kell megoldani. Az élköteg élei között ezért ún. "ÉS" kapcsolat van: hiszen minden részproblémát meg kell oldani.
- egy csúcsból több élköteg is indulhat, ha egy probléma többféleképpen dekomponálható. Ezen élkötegek élei között ún. "VAGY" kapcsolat áll fenn: hiszen választhatunk, hogy melyik élköteg mentén oldjunk meg egy problémát.

5.3 ÉS/VAGY gráfok

1. AZ R=(N,A) élsúlyozott irányított hiper-gráf, ahol az
   -N a csúcsok halmaza
   -A $\subseteq$ { (n,M)$\in$ NxN$^+$ | $0 \neq |M| < \infty$ } 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. $(0<\delta \leq c(n,M))$

5.4 Megoldás-gráf

Az eredeti problémát egyszerű problémákra visszavezető dekomponálási folyamatot az ÉS/VAGY gráf speciális részgráfja, az ún. megoldás-gráf jeleníti meg, amelyben:

-szerepel a startcsúcs

-a startcsúcsból minden más csúcsba vezet út, és minden csúcsból vezet út egy megoldás-gráfbeli célcsúcsba

-egy éllel együtt az összes azzal "ÉS" kapcsolatban álló él is (azaz a teljes élköteg) része a megoldás-gráfnak

-nem tartalmaz "VAGY" kapcsolatban álló él párokat

A megoldás a megoldás-gráfból olvasható ki.

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

Az $n^{\alpha}\rightarrow M$ hiper-út ($n\in N, M\in N^{+}$) egy olyan véges részgráf, amelyben:

-M csúcsaiból nem indul hiper-él

-M-en kívüli csúcsokból csak egy hiper-él indul

-minden csúcs elérhető az n csúcsból egy közönséges irányított úton.

A megoldás-gráf egy olyan hiper-út, amely a startcsúcsból csupa célcsúcsba vezet.

5.6 Hiper-út bejárása

Az $n\rightarrow M$ hiper-út egy bejárásán a hiper-út csúcsaiból képzett sorozatoknak a felsorolását értjük:

-első sorozat: <n>

-a C sorozatot a $C^{k\leftarrow K}$ sorozat követi (ahol a $k\in C$, és k minden C-beli előfordulásának helyén a K sorozat szerepel) feltéve, hogy a hiper-útnak van olyan (k,K) hiper-éle, ahol $k\notin M$.

5.7 Útkeresés ÉS/VAGY gráfban

Amikor a startcsúcsból induló hiper-utakat (ezek között vannak a megoldás-gráfok is, ha egyáltalán vannak ilyenek) a bejárásukkal írjuk le, akkor ezek a bejárások olyan közönséges irányított utak, amelyek csúcsai az eredeti ÉS/VAGY gráf csúcsainak sorozatai. Ezen utakból egy olyan közönséges irányított gráfot készíthetünk, amelyben a startcsúcs az ÉS/VAGY gráf startcsúcsából álló egy elemű sorozat, a célcsúcsait leíró sorozatok pedig kizárólag az ÉS/VAGY gráf célcsúcsainak egy részét tartalmazzák.

Ha ebben a közönséges gráfban megoldási utat találunk, akkor az egyben az eredeti ÉS/VAGY gráf megoldás-gráfja is lesz.

6 Keresések

6.1 KR vezérlési szintjei

Három féle vezérlési stratégiát különböztetünk meg:
-általános (független a feladattól, és annak modelljétől: nem merít sem a feladat ismereteiből, sem a modell sajátosságaiból.)
-modellfüggő (nem függ a feladat ismereteitől, de épít a feladat modelljének általános elemeire.)
-heurisztikus (a feladattól származó, annak modelljében nem rögzített, a megoldást segítő speciális ismeret)

Másik megközelítés alapján, kétféle általános stratégiát különböztetünk meg:
-nemmódosítható (lokális keresések, evolúciós algoritmus, rezolúció)
-módosítható (visszalépéses keresések, gráfkeresések)

6.2 Lokális keresések

A lokális keresés olyan KR, amely a probléma reprezentációs gráfjának egyetlen csúcsát (aktuális csúcs) és annak szűk környezetét tárolja (a globális munkaterületén). Kezdetben az aktuális csúcs a startcsúcs, és a keresés akkor áll le, ha az aktuális csúcs a célcsúcs lesz.
Az aktuális csúcsot minden lépésben annak környezetéből vett "jobb" csúccsal cseréli le (keresési szabály).
A "jobbság" eldöntéséhez (vezérlési stratégia) egy kiértékelő (cél-, rátermettségi-, heurisztikus) függvényt használ, amely reményeink szerint annál jobb értéket ad egy csúcsra, minél közelebb esik az a célhoz.

6.3 Hegymászó algoritmus

Mindig az aktuális (akt) csúcs legjobb gyermekére lép, amelyik lehetőleg nem a szülője.
(Megjegyzés: 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 inkább leáll.)
Hátrányok:
Csak erős heurisztika esetén lesz sikeres: különben "eltéved" (nem talál megoldást), sőt zsákutcába jutva "beragad".
Segíthet, ha:
-véletlenül választott startcsúcsból újra- és újra elindítjuk (random restart local search)
-k darab aktuális csúcs legjobb k darab gyerekére lépünk (local beam search)
-gyengítjük a mohó stratégiáját (simulated annealing)
Lokális optimum hely körül vagy ekvidisztans felületen (azonos értékű szomszédos csúcsok között) található körön, végtelen működésbe eshet.
Segíthet ha:
-növeljük a memóriát (tabu search)

6.4 Tabu keresés

A globális munkaterületén az aktuális csúcson (akt) kívül nyilvántartja még:
-az utolsó néhány érintett csúcsot: Tabu halmaz
-az eddigi legjobb csúcsot: optimális csúcs (opt)

Egy keresési szabály minden lépésben
-az aktuális csúcsnak a legjobb, de nem a Tabu halmazban lévő gyerekére lép
-ha akt jobb, mint az opt, akkor opt az akt lesz
-frissíti akt-tal a sorszerkezetű Tabu halmazt

Terminálási feltételek:
-ha az opt a célcsúcs
-ha az opt sokáig nem változik

Előnyök:
Tabu méreténél rövidebb köröket észleli, és ez segíthet a lokális optimum hely illetve az ekvidisztans felület körüli körök leküzdésében.

Hátrányok: A tabu halmaz méretét kísérletezéssel kell belőni.
Zsákutcába futva nem-módosítható stratégia miatt beragad.

6.5 Szimulált hűtés

A keresési szabály a következő csúcsot véletlenszerűen választja ki az aktuális (akt) csúcs gyermekei közül.
Ha az így kiválasztott új csúcs kiértékelő függvény-értéke nem rosszabb, mint az akt csúcsé (itt f(új) $\leq$ f(akt)), akkor elfogadjuk aktuális csúcsnak.
Ha az új csúcs függvényértéke rosszabb (itt f(új) $>$ f(akt)), akkor egy olyan véletlenített módszert alkalmazunk, ahol az új csúcs elfogadásának valószínűsége fordítottan arányos az | f(akt) - f(új) | különbséggel:
$e^{\frac{f(akt)-f(uj)}{T}} >$ random[0,1]

6.6 Hűtési ütemterv

Egy csúcs elfogadásának valószínűségét az elfogadási képlet kitevőjének T együtthatójával szabályozhatjuk. Ezt egy (T_k,L_k) k=1,2,... ütemterv vezérli, amely L_1, majd L_2 lépésen keresztül T_2, stb. lesz.
$e^{\frac{f(current)-f(new)}{T_k}} >$ rand[0,1]
Ha T_1, T_2, ... szigorúan monoton csökken, akkor egy ugyanannyival rosszabb függvényértékű új csúcsot kezdetben nagyobb valószínűséggel fogad el a keresés, mint később.

6.7 Lokális kereséssel megoldható feladatok

Erős heurisztika nélkül nincs sok esély a cél megtalálására.
-Jó heurisztikára épített kiértékelő függvénnyel elkerülhetőek a zsákutcák, a körök.
A sikerhez az kell, hogy egy lokálisan hozott rossz döntés ne zárja ki a cél megtalálását!
-Ez például egy erősen összefüggő reprezentációs-gráfban automatikusan teljesül, de kifejezetten előnytelen, ha a reprezentációs-gráf egy irányított fa.
(Például az n-királynő problémákat csak tökéletes kiértékelő függvény esetén lehetne lokális kereséssel megoldani.)

6.8 A heurisztika hatása a KR működésére

A heurisztika olyan, a feladathoz kapcsolódó ötlet, amelyet közvetlenül építünk be egy algoritmusba azért, hogy annak eredményessége és hatékonysága javuljon (egyszerre képes javítani a futási időt és a memóriaigényt), habár erre általában semmiféle garanciát nem ad.

7 Visszalépéses keresés

A visszalépéses keresés egy olyan KR, amely:
-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

7.1 Visszalépés feltételei

-Zsákutca: az aktuális csúcsból (azaz az aktuális út végpontjából) nem vezet tovább él
-Zsákutca torkolat: az aktuális csúcsból kivezető utak nem vezettek célba
-Kör: az aktuális csúcs szerepel már korábban is az aktuális úton
-Mélységi korlát: az aktuális út hossza elér egy előre megadott értéket

7.2 Alacsonyabb rendű vezérlési stratégiák

-A vezérlési stratégia kiegészíthető:
  -sorrendi szabállyal: sorrendet ad az aktuális út végpontjából kivezető élek (utak) vizsgálatára
  -vágó szabállyal: megjelöli azokat az aktuális út végpontjából kivezető éleket (utakat), amelyeket nem érdemes megvizsgálni
-Ezek a szabályok lehetnek:
  -másodlagos vezérlési stratégiák (a probléma modelljének sajátosságaiból származó ötlet)
  -heurisztikák (a probléma ismereteire támaszkodó ötlet)

7.3 Első változat: VL1

A visszalépéses algoritmus első változata az, amikor a visszalépés feltételei közül az első kettőt építjük be a kereső rendszerbe.
Bebizonyítható: Véges körmentes irányított gráfokon a VL1 mindig terminál, és ha létezik megoldás, akkor talál egyet. UI: véges sok adott startból induló út van.
Rekurzív algoritmussal (VL1) szokták megadni.

7.4 Az n-királynő probléma új reprezentációs modellje

Az előző módszerek átalakították az n-királynő probléma reprezentációját:
  Tekintsük a D_1,...,D_n halmazokat, ahol D_i=1...n
  (ezek az i-dik sor szabad mezői).
  Keressük azt az $(x_1,...,x_n) \in D_1$ x ... x D_n elhelyezést (x_i az i-dik sorban elhelyezett királynő oszloppozíciója),
  amely nem tartalmaz ütést: minden i,j királynő párra:
  $C_{ij}(x_i,x_j) \equiv (x_i \neq x_j \bigwedge |x_i\text{-}x_j| \neq |\text{i-j}|)$.
A visszalépéses keresés e modell változóinak értékeit határozza meg, miközben a bemutatott vágó módszerek egyike redukálják ezen változók D_i halmazait.

7.5 Bináris korlát-kielégítési modell

Keressük azt az $(x_i,...,x_n) \in D_1$ x...xD_n n-est (D_i véges) amely kielégít néhány $C_{ij} \subseteq D_i$xD_j bináris korlátot.
Példa:
Házasságközvetítő probléma (n férfi, m nő, keressünk minden férfinak neki szimpatikus feleségjelöltet):
  -Az i-dik férfi (i=1..n) felesége (x_i) a D_i=1,...,m azon elemei, amelyekre fenn áll, hogy szimpatikus(i,x_i).
  -Az összes (i,j)-re: $C_{ij}(x_i,x_j) \equiv (x_i,x_j)$ (azaz nincs bigámia)

7.6 Modellfüggő vezérlési stratégia

A korábban mutatott vágó módszereket az új modellben a bináris korlátok definiálják, de a korlátok jelentésétől függetlenül. Ezek a módszerek tehát nem heurisztikák, hanem modellfüggő vágó stratégiák:

Töröl(i,k): $D_i := D_i$- { e$\in D_i$| $\neg$C$_{\text{ik}}$(e,x_k)}

Szűr(i,j) :$D_i := D_i$ - { e$\in D_i$ | $\forall$f$\in D_j$:$\neg$C$_{\text{ij}}$(e,f)}

Modellfüggő sorrendi stratégiák is konstruálhatók:

-Mindig a legkisebb tartományú még kitöltetlen komponensek válasszunk előbb értéket.

- Ugyanazon korláthoz tartozó komponenseket lehetőleg közvetlenül egymás után töltsük ki.

7.7 Második változat: VL2

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

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

Rekurzív algoritmussal (VL2) adjuk meg

1: akt := utolsó_csúcs(út)
2: if cél(akt) then return(nil) endif
3: if hossza(út) $\geq$ korlát then return(hiba) endif 4: if akt $\in$ maradék(út) then return(hiba) endif
5: for $\forall$új $\in \gamma$(akt) - π(akt) loop
6: megoldás:= VL2(fűz(út,új))
7: if megoldás $\neq$ hiba then
8: return(fűz((akt,új),megoldás)) endif
9: endloop
10:return (hiba)

7.8 Mélységi korlát szerepe

A VL2 nm talál megoldást (csak terminál), ha a megadott mélységi korlátnál csak hosszabb megoldási utak vannak.

A mélységi korlát önmagában is biztosítja a terminálást körök esetén is.

- Ez akkor előnyös, ha nincsenek rövid körök (a kettő hosszú köröket kiszűri a szülőcsúcs vizsgálat).

-Ilyenkor nem kell a rekurzív hívásnál a teljes aktuális utat átadni : elég az út hosszát, az aktuális csúcsot és annak szülőjét.

7.9 Értékelés

Előnyök:
-Mindig terminál, talál megoldást (a mélységi korláton belül)
-Könnyen implementálható
-Kicsi memória igény

Hátrányok:
-Nem ad optimális megoldást. (iterációba szervezhető)
-Kezdetben hozott rossz döntést csak sok visszalépés korrigál (visszaugrásos keresés)
-Egy zsákutca részt többször is bejárhat a keresés

8 Gráfkeresés

A gráfkeresés olyan KR, amelynek:
-globális munkaetülete: a reprezentációs gráf startcsúcsból kiinduló már feltárt útjait tárolja (tehát egy részgráfot), és külön az egyes utak végeit, a nyílt csúcsokat
-kiinduló értéke: a startcsúcs,
-terminálási feltétel: megjelenik egy célcsúcs vagy megakad az algoritmus.
-keresés szabálya: egyik útvégi csúcs kiterjesztése
-vezérlés stratégiája: a legkedvezőbb csúcs kiterjesztésére törekszik

8.1 Általános gráfkereső algoritmus

Jelölések:
-keresőgráf (G): a reprezentációs gráf eddig bejárt és eltárolt része
-nyílt csúcsok halmaza (NYÍLT): kiterjesztésre várakozó csúcsok, amelyeknek gyerekeit még nem vagy nem eléggé jól ismerjük.
-kiterjesztett csúcsok halmaza (ZÁRT) : azok a csúcsok, amelyeknek a gyerekeit már előállítottuk.
-kiértékelő függvény (f: NYÍLT $\rightarrow \mathbb{R}$): kiválasztja a megfelelő nyílt csúcsot kiterjesztésre

8.2 Kritika

-Nem olvasható ki a megoldási út a kereső gráfból
-Meg kell jegyezni a felfedezett utak nyomát.
Nem garantál optimális megoldást (sőt még a megoldást sem)
-Tároljuk el egy csúcsnál az odavezető eddig talált legjobb út költségét.
Körökre érzékeny
-Ha ehhez a csúcshoz egy kört tartalmazó utat találunk, akkor annak költsége drágább lesz a tárolt értéknél, hiszen δ-gráfban vagyunk.

8.3 Gráfkeresés függvényei

π(n)= N$\rightarrow$ N szülőre visszamutató pointer
π (n)=n csúcs már ismert szülője, π(start)=nil
π egy start gyökerű irányított feszítőfát jelöl ki G-ben: π feszítőfa, π-út
Jó lenne, ha a π-út optimális start$\rightarrow$n G-beli utat jelölne ki: a π feszítőfa optimális lenne.
g: N$\rightarrow\mathbb{R}$ költség függvény
g(n)=c$^{\alpha}$(start,n) - egy már megtalált $\alpha\in$ start $\rightarrow$n út költsége
Jó lenne ha minden n-re a g(n) a π-út költségét mutatná, azaz a π és g konzisztens lenne.

8.4 A korrektség fenntartása

Kezdetben: π(start) :=nil, g(start) :=0
Az n csúcs kiterjesztése után minden m$\in\gamma$(n) csúcsra

1. Ha m új csúcs
   azaz m$\notin$G akkor
   π(m) :=n, g(m) := g(n)+c(n,m)
   NYÍLT := NYÍLT $\cup$ m
2. Ha m régi csúcs, amelyhez olcsóbb utat találtunk
   azaz m $\in$ G és g(n) + c(n,m) $<$ g(m) akkor
   π(m) :=n, g(m) :=g(n)+c(n,m) //g(n) értéke ekkor csökken
3. Ha m régi csúcs, amelyhez nem találtunk olcsóbb utat
   azaz m$\in$G és g(n) + c(n,m) $\geq$g(m) akkor SKIP

8.5 Általános gráfkereső algoritmus

1: G:= (start, $\emptyset$);NYÍLT:=start;g(start):=0;π(start):=nil
2: loop
3: if empty(NYÍLT) then return nincs megoldás
4: n:=min_f(NYÍLT)
5: if cél(n) then return megoldás
6: NYÍLT :=NYÍLT-n
7: for $\forall$m$\in\delta$(n)-π(n) loop
8: if(m) $\notin$ G or g(n) + c(n,m) $<$g(m) then
9: π(m):=n; g(m):=g(n)+c(n,m); NYÍLT :=NYÍLT$\cup$m
10: endloop
11: G:=G $\cup$ (n,m)$\in$A | m $\in\gamma$(n)-π(n)
12: endloop

8.6 Működés és eredmény

A GK δ-gráfban a működése során egy csúcsot legfeljebb véges sokszor terjeszt ki. (Ebből következik például, hogy körökre nem érzékeny)

A GK véges δ-gráfban mindig terminál.
Ha a véges δ-gráfban létezik megoldás, akkor a GK megoldás megtalálásával terminál.

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

8.7 Nevezetes gráfkereső algoritmusok

Most az f kiértékelő függvény megválasztása következik.

Nem-informált	Heurisztikus
mélységi (MGK)	előre tekintő (mohó, best-first)
szélességi (SZGK)	A, A*, A^c*
egyenletes (EGK)	A**,B

8.8 Heurisztika a gráfkereséseknél

Heurisztikus függvénynek nevezzük azt a h:N$\rightarrow\mathbb{R}$ függvényt, amelyik egy csúcsnál megbecsüli a csúcsból a célba vezető ("hátralévő") optimális út költségét.
$h(n) \approx \min_{t\in T} c^*(n,t) = c^*(n,T) = n^*(n)$ (h^*:N$\rightarrow\mathbb{R}$)
Ez egy az eddiginél szigorúbb definíciója a heurisztikának.

8.9 Heurisztikus függvények tulajdonságai

Nevezetes tulajdonságok:
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)

8.10 A memória igény vizsgálata

$ZART_s$ az S gráfkereső algoritmus által lezárt (kiterjesztett csúcsok halmaza)
Rögzítsünk egy feladatot és két, X ésé Y gráfkereső algoritmust
Az adott feladatra nézve
a. az X nem rosszabb az Y-nál, ha $ZART_X \subseteq ZART_Y$
b. az X jobb az Y-nál, ha $ZART_X \subseteq ZART_Y$
Ezek alapján összevethető
1. két eltérő heurisztikájú A* algoritmus ugyanazon a feladaton, azaz a két

heurisztika.
2. két útkereső algoritmus, például az A* algoritmus és egy másik szintén optimális megoldást garantáló- gráfkereső algoritmus a megengedhető problémák egy részhalmazán.

8.11 Különböző heurisztikájú A* algoritmusok memória igényének összehasonlítása

Az A_1 (h_1 heurisztikával) és A_2 (h_2 heurisztikával) A* algoritmusok közül az A_2 jobban informált, mint az A_1, ha minden n∈N/ T csúcsra teljesül, hogy $h_1(n)<h_2(n)$.
Bebizonyítható, hogy a jobban informált A_2 nem rosszabb a kevésbé informált A_1-nél, azaz $\text{ZÁRT}_{A2} \subseteq \text{ZÁRT}_{A1}$

8.12 A futási idő elemzése

Zárt csúcsok száma: k=|ZÁRT|
Alsókorlát: k
Egy monoton megszorításos heurisztika mellett egy csúcs legfeljebb csak egyszer terjesztődik ki,
habár ettől még a kiterjesztett csúcsok száma igen sok is lehet
(lásd egyenletes keresés)
Felsőkorlát: 2^{k-1}
lásd. Martelli példáját

8.13 B algoritmus

Martelli javasolta belső kiértékelő függvénynek a g költség függvényt.
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(NYÍLT)<F
then n:= min_g(m∈NYÍLT | f=(m)<F)
else n:= min_f(NYÍLT); F:=f(n)
endif elágazással.

8.14 B algoritmus futási ideje

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

Legrosszabb esetben:
minden zárt csúcs először küszöbcsúcsként terjesztődik ki.
(Csökkenő kiértékelő függvény mellett egy csúcs csak egyszer,
a legelső kiterjesztésekor
lehet köszöb.)
Az i-dik árok legfeljebb az összes addigi i-1 darab küszöbcsúcsot
tartalmazhatja (a start csúcs nélkül).

Így az összes kiterjesztések száma legfeljebb $1/2 \text{x} k^2$

8.15 Heurisztika szerepe

Milyen a jó heurisztika?
-megengedhető: $h(n) \leq h^*(n)$
Bár nincs mindig szükség optimális megoldásra.
-jól informált: $h(n) \quad h^*(n)$
-monoton megszorítás: $h(n)-h(m) \leq c(n,m)$
Ilyenkor nem érdemes B algoritmust használni
Változó heurisztikák:
-$f=g+\phi \text{x} h$ ahol ϕ d
-B' algoritmus

if $h(n) < \min_{m \in \gamma(n)}(c(n,m)+h(m))$
then $h(n) := \min_{m \in \gamma(n)}(c(n,m)+h(m))$
else for $\forall m \in \gamma(n)$-re loop
if $h(n)-h(m) > c(n,m)$ then $h(m) := h(n)-c(n,m)$
endloop

A h megengedhető marad
A h nem csökken
A monoton megszorításos élek száma nő

8.16 Mohó A algoritmus

Nincs mindig szükség az optimális megoldásra. Ilnyekor a mohó A* algoritmus is használható, amely rögtön megáll, ha célcsúcs jelenik meg a NYÍLT-ban.
Mohó A* algoritmus csak a megoldás megtalálását garantálja. De belátható:
Ha h megengedhető és $\forall t \in T$: $\forall (n,t) \in A$:$h(n)+\alpha \geq c(n,t)$, akkor a talált megoldás költsége: $g(t) \leq h^*(s)+\alpha$
A mohó A* algoritmus megengedhető heurisztika mellett akkor garantálja az optimális megoldást is,
ha $\forall\ t \in T$: $\forall (n,t) \in A$: $h(n)=c(n,t)$ vagy
ha h monoton és $\exists \alpha \geq 0$:$\forall t \in T$:$\forall (n,t) \in A$:$h(n)+\alpha=c(n,t)$

9 Kétszemélyes játékok

9.1 Kétszemélyees, teljes információjú, véges determinisztikus, zéró összegű játékok

Két játékos lép felváltva adott szabályok szerint, amíga játszma vége nem ér.
Mindkét játékos ismeri a maga és az ellenfele összes múltbeli és jövőbeli lépéseit és lépési lehetőségeit, és azok következményeit.
Minden lépés véges számú lehetőség közül választható, és minden játszma véges lépésben véget ér. Egy lépés determinisztikus, a véletlennek nincs szerepe.
Amennyit a játszma végén az egyik játékos nyer, annyit veszít a másik. (Legegyszerűbb változatban két esélyes: egyik nyer, másik veszít; vagy három esélyes: döntetlen is megengedett)

9.2 Állapottér-reprezentáció

állapot -állás + soron következő játékos
művelet -lépés
kezdő állapot -kezdőállás + kezdő játékos
végállapot -végállás + játékos
payoff függvény: p_A,p_B: végállapot $\rightarrow \mathbb{R}$ (játékosok: A,B)
Zéró összegű kétszemélyes játékban:
$p_A(t) + p_B(t)=0$ minden t végállapotra
Speciális esetben:
$p_A(t)=+1$ ha A nyer
$p_A(t)=-1$ ha A nyer
$p_A(t)=0$ ha A nyer

9.3 Játékfa

csúcs: állás(egy állás több csúcs is lehet)
szint: játékos(felváltva az A és B szintjei)
él: lépés (szintről szintre)
gyökér: kezdőállás(kezdő játékos)
levél: végállások
ág: játszma

9.4 Nyerő stratégia

Egy játékos nyerő stratégiája egy olyan elv, amelyet betartva az ellenfél minden lépésére tud olyan választ 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.
Hasznos lehet a nem-vesztő stratégia megtalálása is, ha döntetlent is megengedő játéknál nincs győztes stratégia.
Általános zéró összegű játékoknál beszélhetünk adott hasznosságot biztosító stratégiáról. TÉTEL: A két esélyes (győzelem vagy vereség) teljes információjú véges determinisztikus kétszemélyes játékokban az egyik játékos számára biztosan létezik nyerő stratégia.
A három esélyes játékokban (van döntetlen is) a nem vesztő stratégiát lehet biztosan garantálni.

9.5 Részleges játékfa-kiértékelés

A nyerő vagy nem-vesztő stratégia megkeresése egy nagyobb játékfa esetében reménytelen.
Az optimális lépés helyett a soron következő jó lépést keressük. Legyen a bennünket képviselő játékos neve mostantól MAX az ellenfélé pedig MIN.
Ehhez az aktuális állapotból indulva kell a játékfa néhány szintjét felépíteni, ezen a részfa leveleinek számunkra való hasznosságát megbecsülni, majd ez alapján a soron következő lépést meghatározni.

9.6 Kiértékelő függvény

Minden esetben szükségünk van egy olyan heurisztikára, amely a mi szempontunkból becsüli meg egy állás hasznosságát: f: Állások $\Rightarrow$ [-1000, 1000] függvény.
Példák:
Sakk: (kiértékelő függvény a fehérnek)
f(s) = (fehér királynő száma) - (fekete királynő száma)

Tic-tac-toe: f(s) = M(s) -O(s)
M(s) = a saját lehetőséges győztes vonalaink száma
O(s) = az ellenfél lehetséges győztes vonalaink száma

9.7 Minimax algoritmus

A játékfának az adott állás csúcsából leágazó részfáját felépítjük néhány szintig
A részfa leveleit kiértékeljük a kiértékelő függvény segítségével.
Az értékeket felfuttatjuk a fában:
A saját (MAX) szintek csúcsaihoz azok gyermekeinek maximumát:
szülő:= max($gyerek_1$,...,$gyerek_k$)
Az ellenfél (MIN) csúcsaihoz azok gyermekeinek minimumát:
szülő:=min($gyerek_1$,...,$gyerek_k$)
Soron következő lépésünk ahhoz az álláshoz vezet, ahonnan a gyökérhez felkerült a legnagyobb érték.

9.8 Átlagoló kiértékelés

Célja a kiértékelő függvény esetleges tévedéseinek simítása.
MAX szintjeire az m darab legnagyobb értékű gyerek (max_m) átlaga, a MIN-re az n darab legkisebb értékű gyerek (min_n) átlaga kerül.

9.9 Váltakozó mélységű kiértékelés

Célja, hogy a kiértékelő függvény minden ágon reális értéket mutasson. Megtévesztő lehet egy csúcsnál ez az érték ha annak szülőjénél a kiértékelő függvény lényegesen eltérő értéket mutat: a játék ezen szakasza nincs nyugalomban.
Egy adott szintig (minimális mélység) mindenképpen felépítjük a részfát,
majd ettől a szinttől kezdve egy adott szintig (maximális mélység) csak azon csúcsokat terjesztjük ki, amelyekre nem teljesül a nyugalmi teszt:
|f(szülő) - f(csúcs) | <K,

9.10 Szelektív kiértékelés

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

9.11 Negamax algoritmus

Negamax ejárást könnyebb implementálni.
Kezdetben (-1)-gyel szorozzuk azon levélcsúcsok értékeit, amelyek az ellenfél (MIN) szintjein vannak, majd
Az értékek felfuttatásánál minden szinten az alábbi módon számoljuk a belső csúcsok értékeit:
szülő := max(-$gyerek_1$,...,-$gyerek_k$)

9.12 Alfa-béta algoritmus

Visszalépéses algoritmus segítségével járjuk be a részfát (olyan mélységi bejárás, amely mindig csak egy utat tárol). Az aktuális úton fekvő csúcsok ideiglenes értékei:
- a MAX szintjein α érték: ennél rosszabb értékű állásba innen már nem juthatunk
- a MIN szintejin 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 $\alpha \geq \beta$.

9.13 Elemzés

Ugyanazt a kezdőlépést kapjuk eredményül, amit a minimax algoritmus talál. (Több egyforma kezdőirány esetén a "baloldalit" választjuk.)
Memória igény: csak egy utat tárol.
Futási idő: a vágások miatt sokkal jobb, mint a minimax módszeré.
Átlagos eset: egy csúcs alatt, két belőle kiinduló ág megvizsgálása után már vághatunk.
Optimális eset: egy d mélységű b elágazású fában kiértékelt levélcsúcsok száma: $\sqrt{b^{\mathrm{d}}}$
Jó eset: A részfa megfelelő rendezésével érhető el.

9.14 Kétszemélyes játékot játszó program

Váltakozó mélységű, szelektív, (m,n) átlagoló, negamax alfa-béta kiértékelést végez.
Keretprogram, amely váltakozva fogadja a felhasználó lépéseit, és generálja a számítógép lépéseit.
Kiegészítő funkciók (beállítások, útmutató, segítség, korábbi lépések tárolása, mentés stb.)
Felhasználói felület, grafika
Heurisztika megválasztása (kiértékelő függvény, szelekció, kiértékelés sorrendje)

10 Evolúciós algoritmusok

10.1 Evolúció, mint kereső rendszer

A problémára adható néhány lehetséges választ, azaz a problématér több egyedét tároljuk egyszerre. Ez a populáció.
Kezdetben egy többnyire véletlen populációt választunk. A cél egy bizonyos célegyed 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 rátermettebbekhez hasonló egyedekre cseréljük le. Ez a változtatás visszavonhatatlan. Ez egy nem-módosítható stratégiájú keresés.

10.2 Evolúciós operátorok és a terminálási feltétel

Szelekció: Kiválasztunk néhány (lehetőleg rátermett) egyedet szülőnek.
Rekombináció (keresztezés): 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.
Terminálási feltétel:
-ha a célegyed megjelenik a populációban.
-ha a populáció egyesített rátermettségi függvény értéke egy ideje nem változik.

10.3 Evolúció alapalgoritmusa

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

10.4 Kielégíthetőségi probléma (SAT)

Adott egy n változós Boolean formula KNF alakban. A változók milyen igazság kiértékelése mellett lesz a formula igaz?
Példa: $(x_1 \vee !x_2 \vee x_5) \wedge (x_1 \vee !x_3) \wedge (!x_1 \vee x_4) \wedge (!x_2 \vee x_5)$
Egy megoldás: x_1=true; x_2=false; x_3=false; x_4=true; x_5=true
Egyed: egy lehetséges igazság kiértékelés
Reprezentáció: logikai érték (bitek) sorozata
Rátermettségi függvény: Az adott formula igazra értékelt klózainak száma

10.5 Evolúciós algoritmus elemei

Problématér egyedeinek reprezentációja: kódolás
Rátermettségi függvény (fitnesz függvény) -kapcsolat a kódolással és a céllal
Evolúciós operátorok - szelekció, rekombináció, mutáció, visszahelyezés
Kezdő populáció, megállási feltétel (cél)
Stratégiai paraméterek - populáció mérete, mutáció valószínűsége, utódképzési ráta, visszahelyezési ráta, stb.

10.6 Kódolás

Egy egyedet egy jelsorozattal (kromoszómával) kódolunk. A jelsorozatnak néha ki kell elégítenie egy kód-invariánst.
Az egyedeket az őket reprezentáló kódjukon keresztül változtatjuk meg. Egy jel vagy jelcsoport, azaz a gén írja le az egyed egy tulajdonságát (attribútum-érték párját).
Sokszor egy génnek a kódsorozatban elfoglalt pozíciója (lókusza) jelöli ki a gén által leírt attribútumot, amelynek értéke maga a gén (allél). A kód ekkor tulajdonságonként feldarabolható: egy rövid kódszakasz megváltoztatása kis mértékben változtat az egyeden.
Gyakori megoldások:
-Vektor: valós vagy egész számok rögzített hosszú tömbje
-Bináris kód: bitek rögzített hosszú tömbje
-Véges sok elem permutációja

10.7 Szelekció

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

10.8 Rekombináció

A feladata az, hogy adott szülő-egyedekből olyan utódokat hozzon létre, amelyek a szüleik tulajdonságait "öröklik".
-Keresztezés: véletlen kiválasztott pozíción jelcsoportok (gének) vagy jelek cseréje.
-Rekombináció: a szülő egyedek megfelelő jeleinek kombinálásával kapjuk az

utód megfelelő jelét.
Ügyelni kell a kód-invariáns megtartására: vizsgálni kell, hogy az új kód értelmes lesz-e (permutáció)

10.9 Permutációk keresztezése

Parciálisan illesztet keresztezés:
Egy szakasz cseréje után párba állítja és kicseréli azokat a szakaszon kívüli elemeket, amelyek megsértik a permutáció tulajdonságot.

Ciklikus keresztezés:
1. Választ egy véletlen i∈[1..length]-t
2. $a_i \leftrightarrow b_i$
3. Keres olyan j∈[1..length]-t (j≠i), amelyre a_j=a_i,
4. Ha nem talál, akkor vége, különben i=j
5. goto 2.

10.10 Rekombináció vektorra

Köztes rekombináció:
-A szülők (x,y) által kifeszített hipertégla környezetében lesz utód (u).
-∀i=1..n:u_i =$a_i x_i$ + (1-a_i)y_i a_i∈[-h, 1+h] véletlen

Lineáris rekombináció:
-A szülők (x,y) által kifeszített egyenesen a szülők környezetében vagy a szülők között lesz az utód (u).
-∀i=1..n: u_i=ax_i + (1-a)y_i a∈[-h, 1+h] véletlen

10.11 Mutáció

A mutáció egy egyed (utód) kis mértékű véletlen változtatását végzi.
Valós tömbbel való kódolásnál kis p valószínűséggel:
-∀i=1..n:z_i=x_i± $range_i$* (1-2*p)
Bináris tömbbel való kódolásnál kis p valószínűséggel: -∀i=1..n:z_i=1-x_i if random[0..1]¡p
Permutáció esetén
-egy jelpár cseréje
-egy kódszakaszon a jelek ciklikus léptetése vagy megfordítása vagy átrendezése.

10.12 Visszahelyezés

A visszahelyezés a populációnak az utódokkal történő frissítése: Kiválasztja a populációnak a lecserélendő egyedeit, és azok helyére a kiválasztott [két szelekció is kell] utódokat teszi.
utódképzési ráta (u)= $\frac{utodokszama}{populacioszama}$
visszahelyezési ráta (v) = $\frac{lecserelendoegyedekszama}{populacioszama}$
-ha u=v, akkor feltétlen cseréről van szó
-ha u<v, akkor egy utód több[további szelekció] példánya is bekerülhet
-ha u>v, akkor az utódok közül[további szelekció] szelektál