Informált keresés

Bevezetés

- **Keresés: probléma megoldása**
- **Nem informált keresés: csak a probléma definícióját ismerik (múlt óra) – csak azt tudjuk, honnan jövünk**
- **Minden szomszédos csomópont egyformán fontosnak számított**
- **Informált keresés: egyéb információval is rendelkeznek (merre lehet a megoldás) – mai óra**

Informált keresés

- **Problémaspecifikus tudást is használunk**
- **Heurisztika: minden állapotban megbecsüli, hogy mekkora az optimális út költsége a(z egyik) célállapotba**
- **h(n): heurisztikus függvény – optimális költség közelítése n állapotból a célállapotba**
- **az *n* állapothoz megad egy becsült értéket a további várható költségről**

Informált keresési módszerek

- g(n): optimális költség a kezdőállapotból n-be
- az eddigi költség *g(n)*
- kiértékelő függvény *f(n)* = *g(n)* + *h(n)* → sorbaállító függvény

Best First/legjobbat először keresés algoritmusa

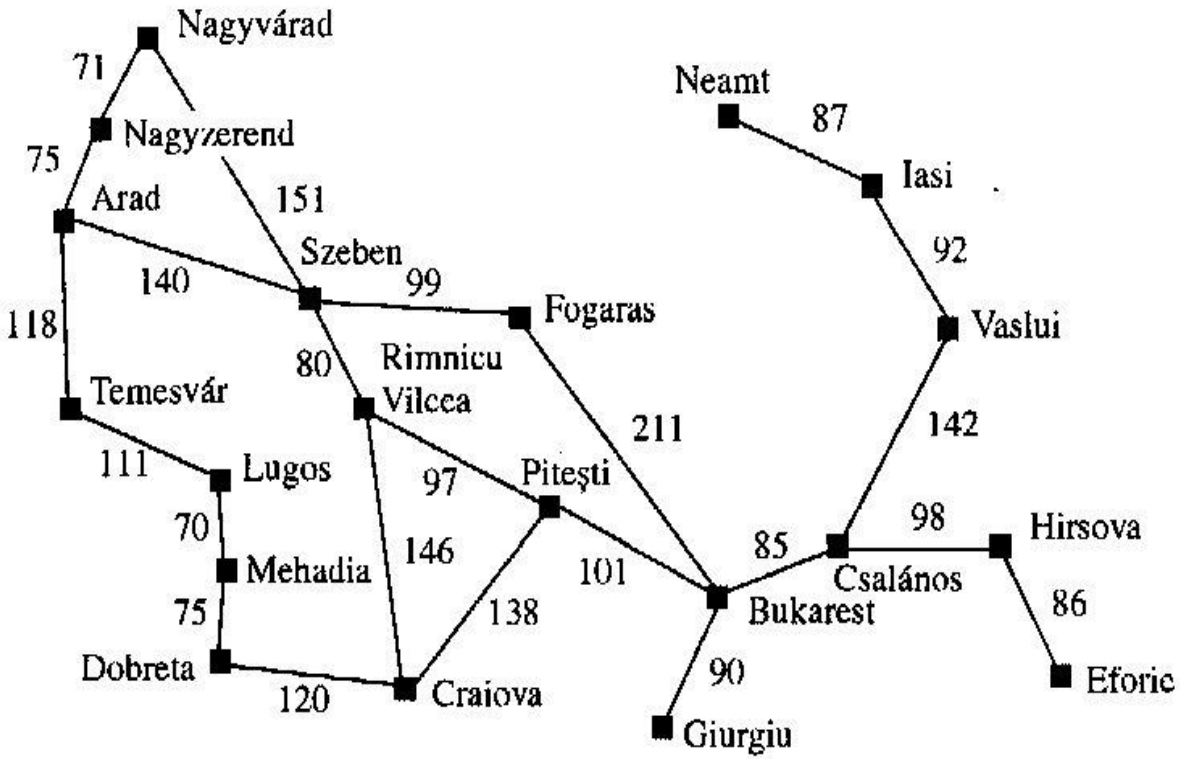
1. Legyen L kezdeti állapotokat tartalmazó lista;
2. Ha L üres, akkor megáll az algoritmus,
3. L minden eleméhez hozzárendelünk egy értéket (f(n)), majd a legkisebb értékű csomópontot kiválasztjuk, legyen ennek neve : n;
4. Ha n a cél, akkor kész, a hozzávezető úttal megadja a megoldást;
5. Egyébként:
   - töröld n-t L-ből;
   - állítsd elő n utódait;
   - jegyezd fel a hozzájuk vezető utat;
   - add az utódokat L-hez;
   - menj 2-re;

Mohó keresés

- **Hasonló a mélységi kereséshez**
- **Minden lépésben a legjobbnak tűnő megoldást választja (legközelebb levő)**
- **h(n)-hez legközelebb levőt választjuk**

Útiköltség és heurisztika

Heurisztika: Bukaresttől légvonalban mért távolság



Bukaresttől légvonalban mért távolság	
Arad	366
Bukarest	0
Craiova	160
Csalános	80
Dobreta	242
Eforie	161
Fogaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugos	244
Mehadia	241
Neamt	234
Nagyvárad	380
Piteşti	98
Rimnicu Vilcea	193
Szeben	253
Temesvár	329
Vaslui	199
Nagyzerend	374

Mohó keresés tulajdonságai

- **nem optimális**
- **nem teljes**
- **időigény b^m (m: maximális mélység)**
- **tárigény b^m (mert az összes csomópontot a tárban tartja)**

A* keresés

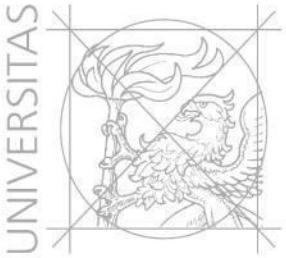
- f(n) = g(n) + h(n)
- Elfogadható heurisztika: nem becsüli felül a célba jutás költségét
- Légvonal: elfogadható
- Ha *h* elfogadható, akkor *f(n)* soha nem becsüli túl az *n* csomóponton keresztül vezető legjobb megoldás valódi költségét

Monotonitás

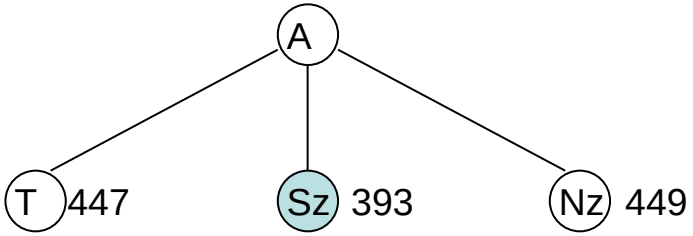
- ***Monoton heurisztikus függvény*: A gyökérből indulva egyetlen út f értéke sem csökken**
- **Elsőnek követett út lesz a legjobb**
- **Háromszög-egyenlőtlenség: a háromszög egyik oldala sem hosszabb, mint a másik két oldal összege**
- **A* optimális, ha h(n) monoton**

L_0: A

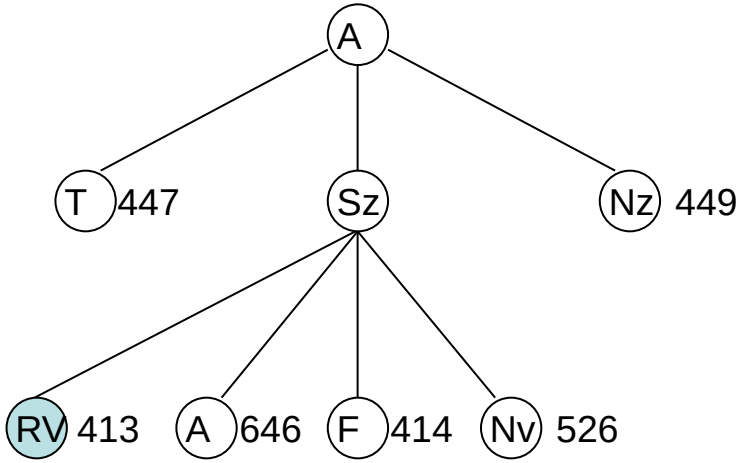
A 366



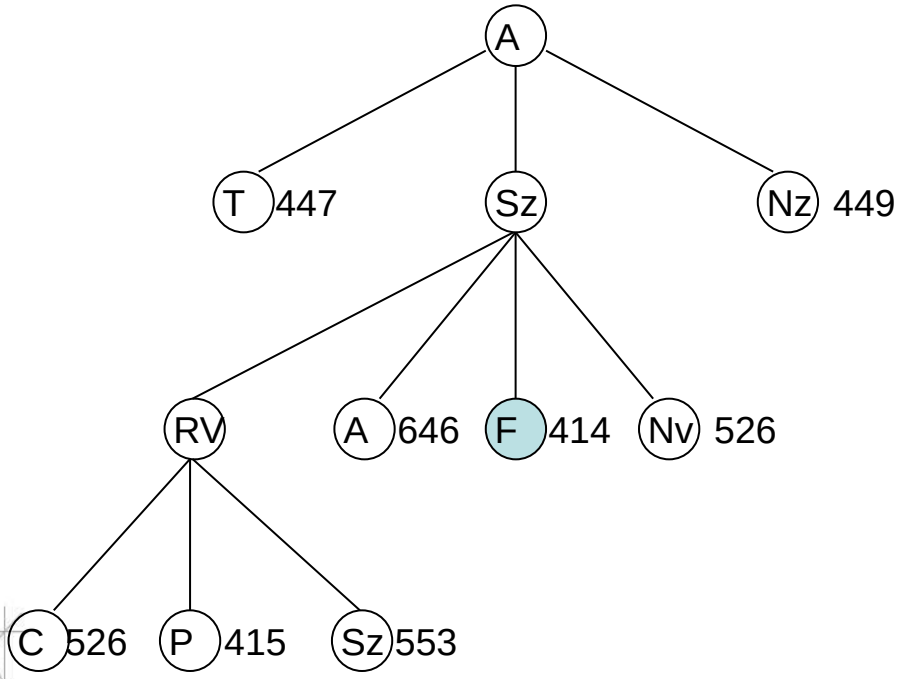
L_1: Sz T Nz



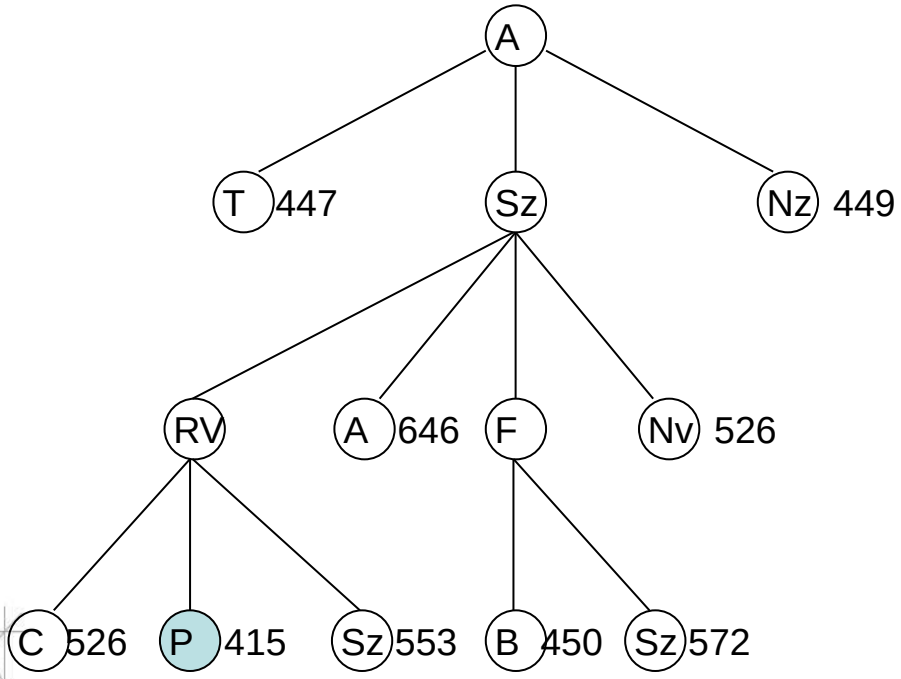
L_2: Rv F T Nz Nv A



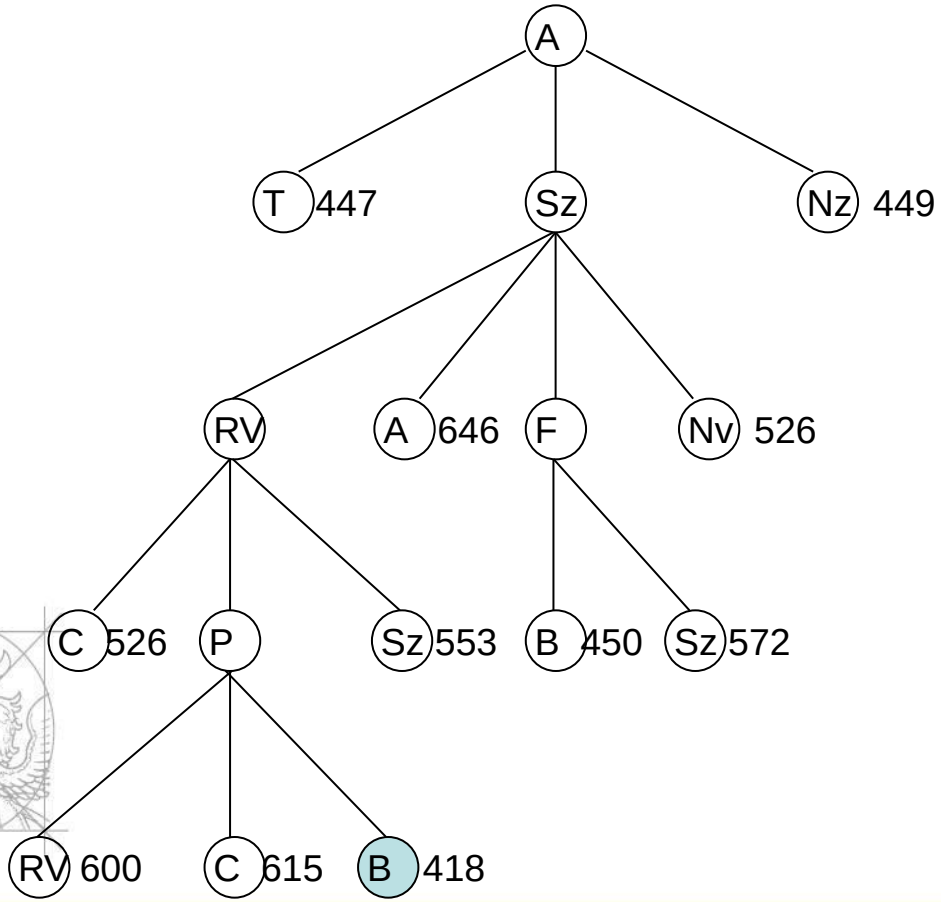
L_3: F P T Nz C Nv Sz A



L_4: P T Nz B C Nv Sz Sz A

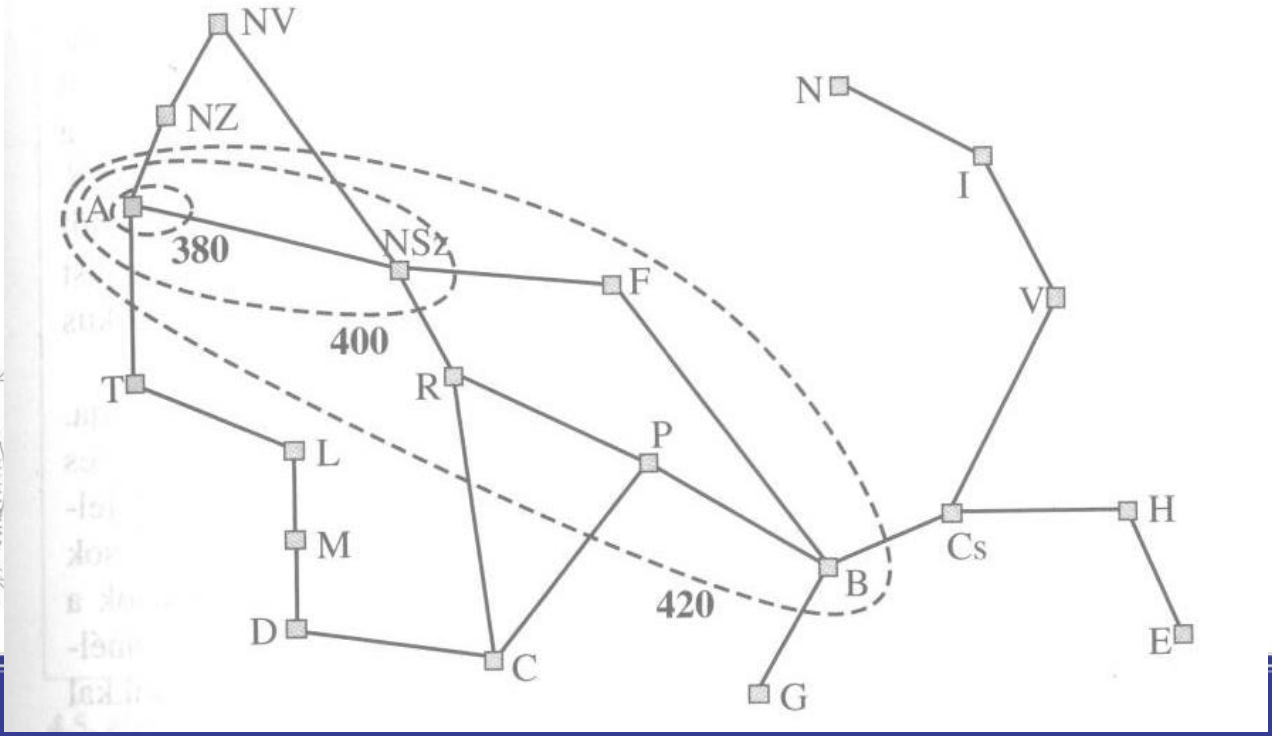


L_5: B T Nz B C Nv Sz Sz Rv C A



Határvonalak

- **Ha f(n) sosem csökken, határvonalak húzhatók be**
- **Nyesés: figyelmen kívül hagyható területek levágása**



A* keresés tulajdonságai

- 1. *f(n)* a legolcsóbb, az *n* csomóponton keresztül vezető megoldás útiköltségének becslője.
- 2. *Elfogadható heurisztika*: a *h(n)* függvény ne becsülje túl a cél eléréséhez szükséges költséget. Ha *h* elfogadható, akkor *f(n)* soha nem becsüli túl az *n* csomóponton keresztül vezető legjobb megoldás valódi költségét.
- 3. *Monoton heurisztikus függvény*: A gyökérből indulva egyetlen út f értéke sem csökken.
- 4. *h(n)=0*, ha n célállapot.

Ha a használt heurisztika ilyen, akkor az algoritmus optimális és teljes.

Memóriakorlátos heurisztikus keresés

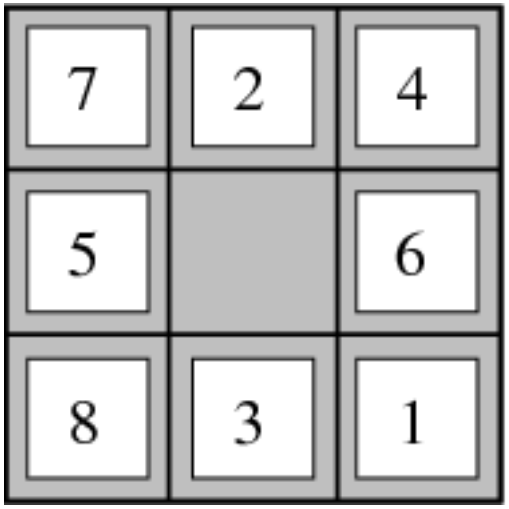
- A* memóriaigénye túl nagy – csökkentsük
- Iteratívan mélyülő algoritmus adaptálása
- Iteratívan mélyülő A* keresés (IDA*)
- Mélységi korlát helyett jósági korlátot alkalmazunk f(n)-re
- Ciklusonként mélységben először keresés
- Mozgó korlát
- Mintha a jósági korlát alatt nem lennének leszármazottak
- Ha a célt nem éri el, újraindítja új korláttal

1. Legyen a mélységi korlát c=0;
2. Legyen L a kezdeti állapotokat tartalmazó lista, és legyen c'=∞ a következő iterációs korlát;
3. Legyen n az első csomópont L-ből:
Ha L=Ø és c'= ∞, akkor nincs megoldás.
Ha L=Ø és c'!= ∞, akkor c=c' és folytasd az algoritmust a 2-es lépéssel!
4. Ha n célállapot, akkor állj és add vissza a hozzátartozó úttal együtt;
5. Egyébként:
Töröld n-et L-ből;
Állítsd elő n valamennyi n' gyermekét;
Minden n'-re:
Ha f(n')<=c, akkor jegyezd fel n'-höz vezető utat;
Add n'-t L elejéhez;
különben
Legyen c'=min(c',f(n')), a következő iteráció mélységi korlátját számítjuk ki;
Menj 3-ra!

Heurisztikus függvények

- **Hogyan kaphatunk jó heurisztikákat?**
- **Relaxáció: feltételek elhagyása**
- **Csak a fontos dolgokat vesszük figyelembe**

Kirakó



- h_1: rossz helyen levő számok
- $h_1() = 8$
- Relaxáció: minden szám egyből a helyére tolható
- h_2: Manhattan-távolság számonként (milyen messze vagyunk a jó helytől)
- $h_2() = 18$
- Relaxáció: minden szám a szomszédba tolható
- $h_{opt}() = 26$

- Nem becsüljük túl a megoldás költségét

Feltételek

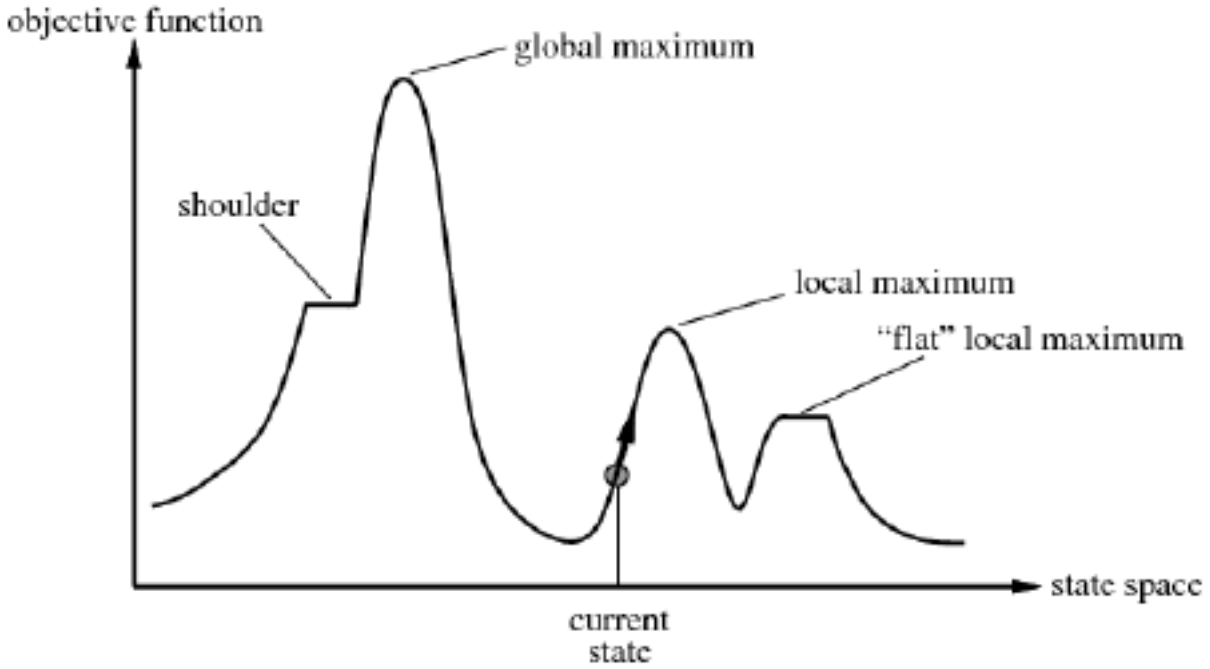
- **A) egy számot csak szomszédos pozícióba tudunk mozgatni**
- **B) egy számot csak üres pozícióba tudunk mozgatni**
- **Mi történik, ha feladjuk a feltételeket?**

Mintaadatbázisok

- **Heurisztika: részproblémák definiálása**
- **Több részproblémára bontjuk a feladatot**
- **Pl. 1,2,3,4 számok kirakása**
- **Megoldások adatbázisban tárolhatók**
- **Független részproblémák: a költségek összeadhatók**

Lokális keresés

- Nem számít a célhoz vezető út
- Költség a fontos
- Globális optimalizálás
- Lehetséges állapotok
- Lehetséges operátorok (cselekvések)
- Állapotátmenet függvény
- Célfüggvény (értéket rendel az állapotokhoz)
- Célállapotok halmaza (globális maximumhelyek)


objective function
global maximum
shoulder
local maximum
"flat" local maximum
state space
current
state

Hegymászó keresés

Algoritmus:

1. Legyen n a kezdeti állapot;

2. Ha n egy cél, akkor állj le és add vissza a megoldást;

3. Egyébként:

- **Állítsd elő n valamennyi n' leszármazottját;**
- **Legyen n a legjobb n';**
- **Menj 2-re!**

Problémák

- **lokális magaslat: „beragad” az algoritmus**
- **Fennsík: lehet-e oldalra menni? (azonos értékű szomszéd felé)**
- **Hegygerinc: lokális maximumok egymás mellett**

Javítások

- **Sztochasztikus hegymászó: véletlen szomszéd azok közül, amelyek jobbak – lassabb, de kevésbé akad el**
- **Első választásos: az első jobb szomszédot választjuk**
- **Véletlen újraindított: ha nem sikeres, véletlen kezdőpontból újraindítjuk**

Szimulált hűtés

- **Fizika: fémet magas hőmérsékletre felmelegítjük, majd fokozatosan lehűtjük (kristályos állapot elérése)**
- **Legmélyebb szakadékba akarjuk juttatni a pingponglabdát**
- **„megrázzuk az asztalt”**

1. Legyen n a kezdeti állapot;

2. Ha n egy cél, akkor állj le és add vissza a megoldást;

3. Egyébként:

Válaszd ki véletlenszerűen n valamelyik n' leszármazottját;

Ha a lépés javít a pillanatnyi helyzeten, akkor azt elfogadja és megteszi, de bizonyos valószínűséggel elfogad olyan lépést is, amely ront a pillanatnyi helyzeten

Legyen n = n';

Menj 2-re!

Lokális nyaláb keresés

- **Nem egy, hanem k állapotot figyelünk**

1 aktuális[] <- K véletlen állapot

2 generáljuk mind a K állapot összes szomszédját

3 aktuális[] <- K legjobb az összes szomszéd közül

4 if aktuális[i] célállapot valamely i-re return aktuális[i]

5 goto 2

Genetikus algoritmusok

- **Populáció: k véletlen módon generált állapot**
- **Utódállapot a két szülőállapot összekombinálásával jön létre**
- **Az utódállapotokat a fittnessfüggvény rangsorolja**
- **Mutáció**

1 aktuális[] <- K véletlen állapot

2 szülők választása

3 új állapotok generálása a szülőkből rekombinációval

4 új állapotok mutációja

5 K állapot kiválasztása a régiek és az újak uniójából

6 if megállási feltétel return

7 goto 2