Diszkrét Irányítások tervezése

Heurisztika

Dr. Bécsi Tamás

Algoritmusok futásideje

- Az algoritmus futásideje függ az N bemenő paramétertől. Azonos feladat különböző N értékek esetén más futásidőt igényelnek. Például:
  - Lineáris keresés esetén az N növekedésével egyenes arányosan nő a futásidő
  - Bináris keresés esetén N növekedésével logaritmikusan nő a futásigény
  - Egyszerű (mondjuk buborék) sorbarendezés esetén N-nel négyzetesen arányosan nő a futásidő

Függvények növekedése

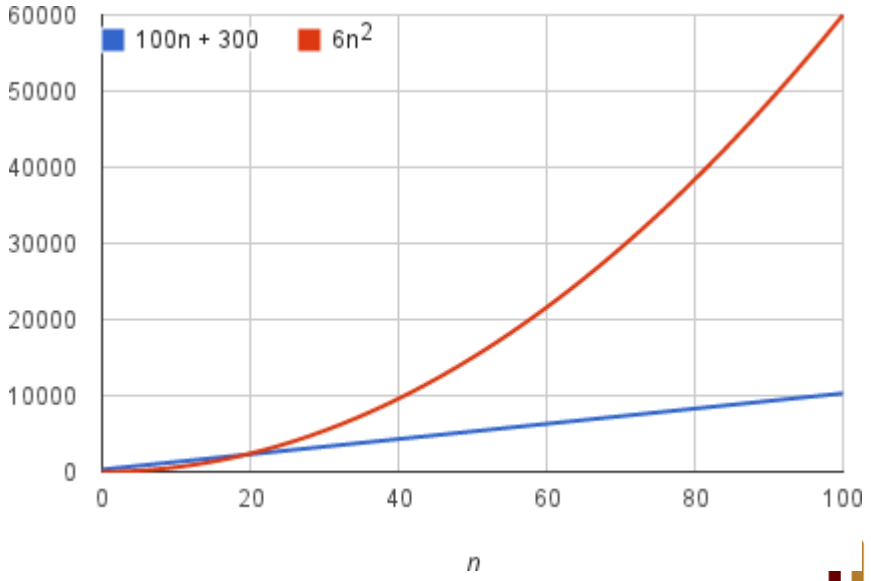
- Ha a bemenet mérete elég nagy, akkor az algoritmus futási idejének csak a nagyságrendje lényeges, ezért az algoritmusnak az ***aszimptotikus*** hatékonyságát vizsgáljuk.
- Ekkor csak azzal foglalkozunk, hogy a bemenet növekedésével miként növekszik az algoritmus futási ideje *határértékben,* ha a bemenet mérete minden határon túl nő. Általában az aszimptotikusan hatékonyabb algoritmus lesz a legjobb választás, kivéve a kis bemenetek esetét.

Aszimptotikus jelölések

- Egy algoritmus aszimptotikus futási idejét leíró jelöléseket olyan függvényekként definiáljuk, melyeknek értelmezési tartománya a természetes számok N = {0, 1, 2, . . .} halmaza.
- Ilyen jelölésekkel kényelmesen leírható a futási idő a legrosszabb esetben, azaz a *T*(N) függvény, amely általában csak egész számú bemenő adattól függ. Mindemellett néha kényelmes az aszimptotikus jelölések többféle *pontatlan* használata.

Példa

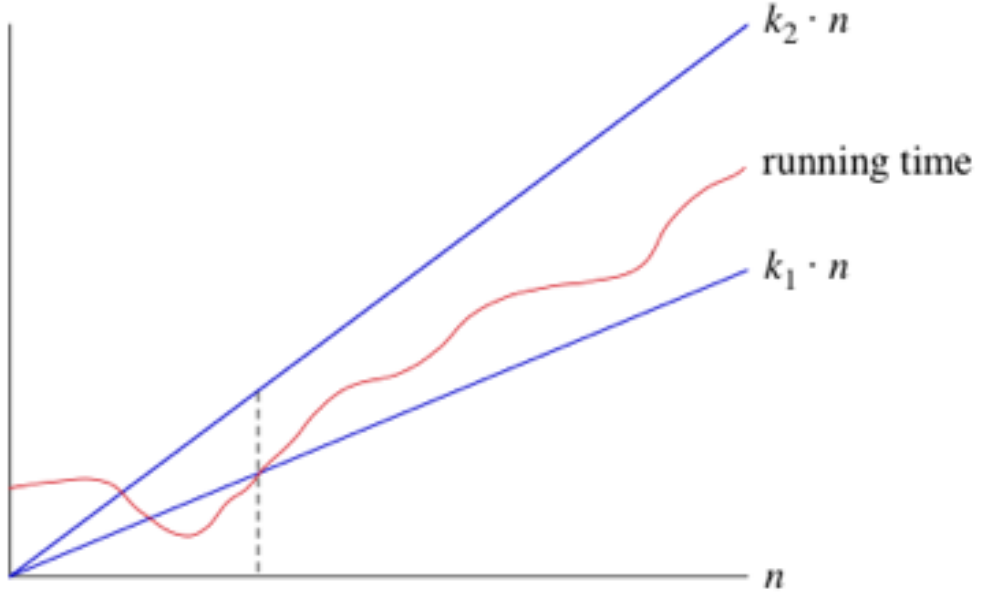
Tételezzük fel, hogy az algoritmusunk futásideje meghatározható, és n darabszám függvényében $T(n) = 6n^2 + 100n + 300$ utasítást igényel. Látható, hogy n>20 esetén a $6n^2$ tag jóval nagyobb, és gyorsabban nő, mint a másik tag. Ekkor az mondjuk, hogy hogy az algoritmus négyzetesen nő. Ez a viselkedés az együtthatóktól független, mindig lesz olyan határszám, ahonnan a négyzetes tag a meghatározó.



Θ jelölés

Definíció: Egy adott g(n) esetén $\Theta(g(n))$-nel jelöljük a függvényeknek azt a halmazát, amelyre:

$\Theta(g(n))$ ={f(n), ha létezik olyan k_1, k_2 és n_0 pozitív állandó, hogy $0 \leq k_1 g(n) \leq f(n) \leq k_2 g(n)$, minden $n > n_0$ esetén}



O jelölés

Definíció: Egy adott g(n) esetén $O(g(n))$-nel jelöljük a függvényeknek azt a halmazát, amelyre:

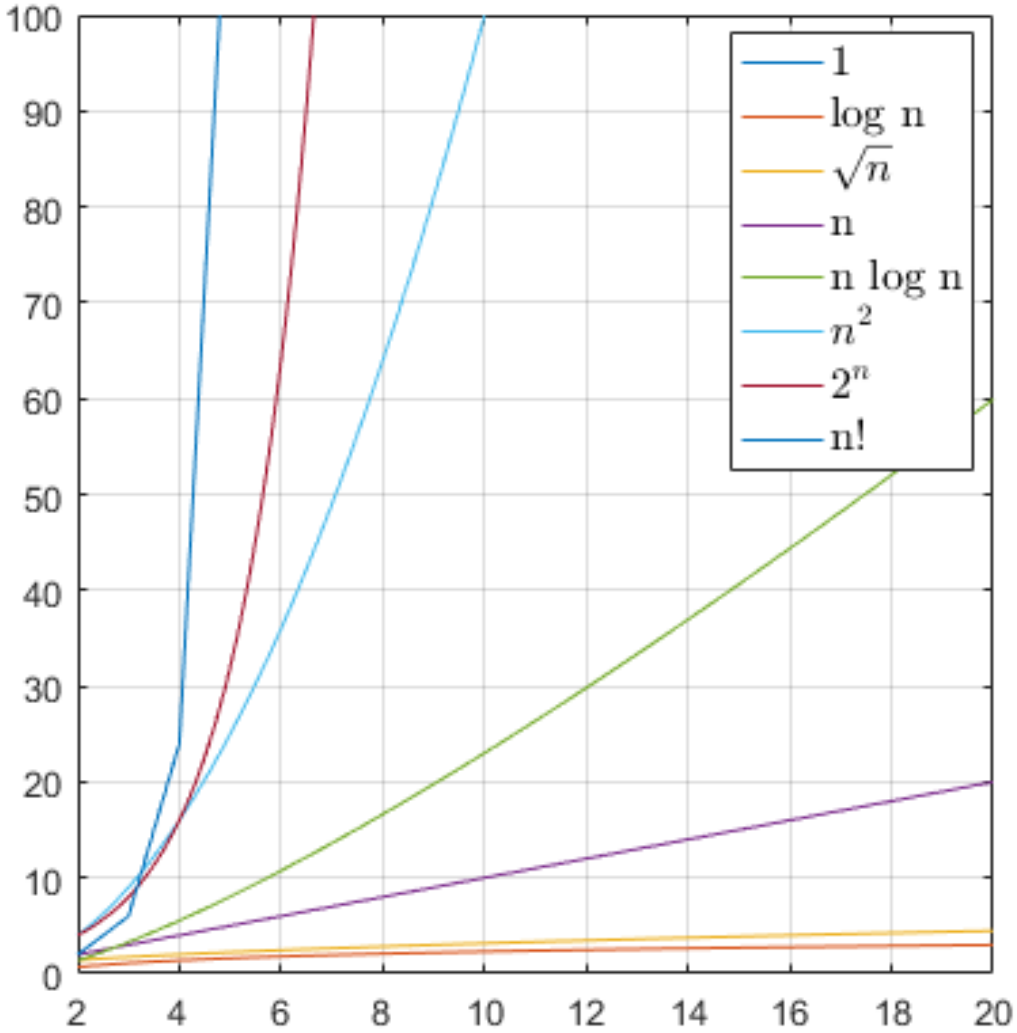
$O(g(n))$ ={f(n), ha létezik olyan k $és$ n_0 pozitív állandó, hogy $0 \leq f(n) \leq kg(n)$, minden $n > n_0$ esetén}

Felső korlát

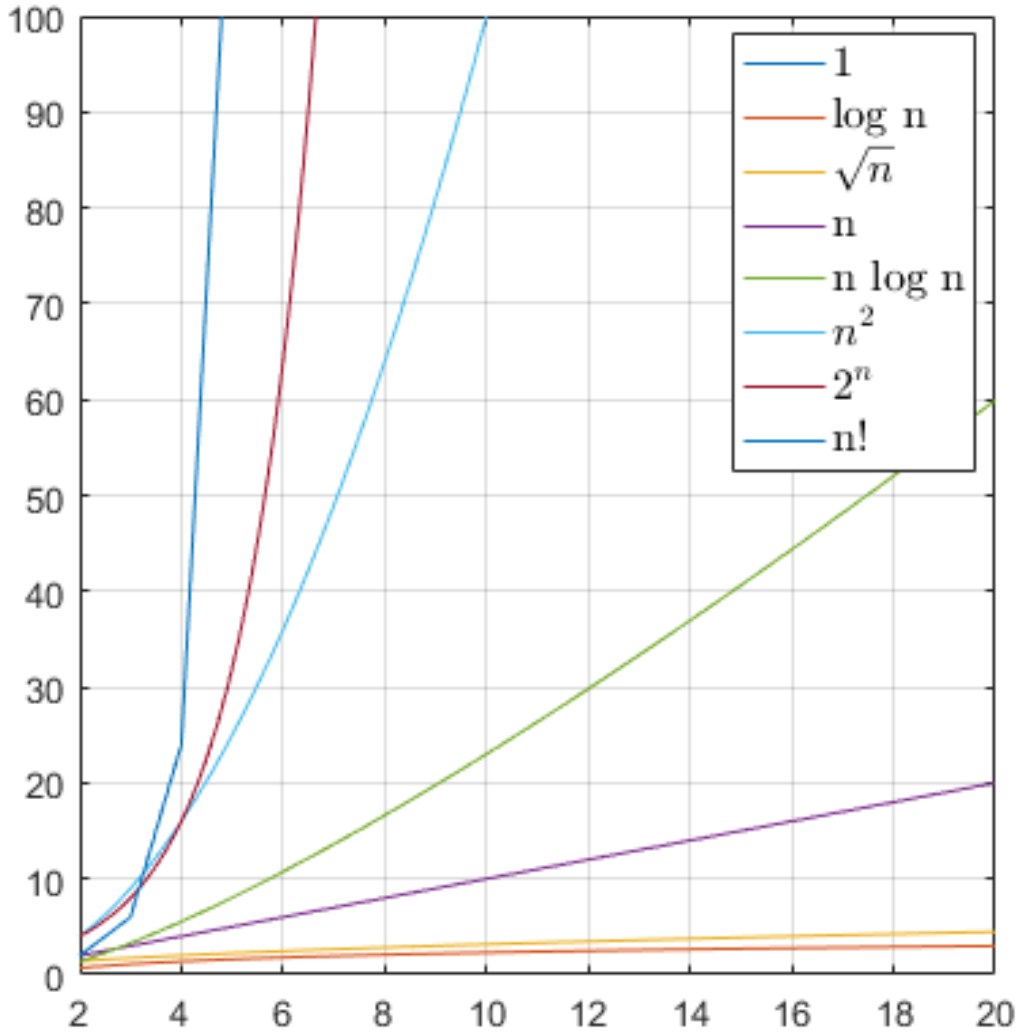
- Értelemszerűen:
  - ha $f(n) = \Theta(g(n))$, akkor $f(n) = O(g(n))$

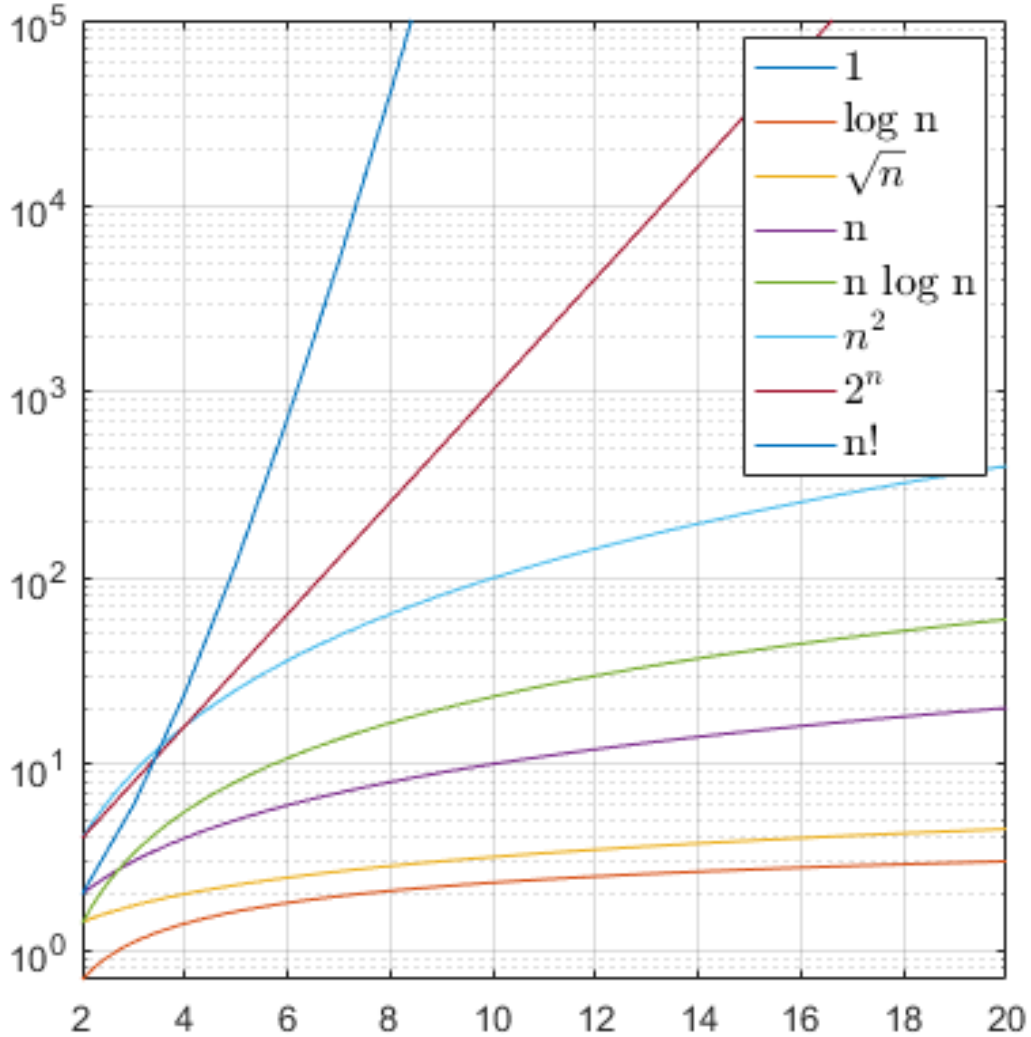
Nagyságrendek, példák

Komp.	Elnevezés	Példa
Θ(1)	konstans	Párosság eldöntése
Θ(lg n)	logaritmikus	bináris keresés
Θ($\sqrt{n}$)	gyökös	példa gyakorlatról
Θ(n)	lineáris	lineáris keresés
Θ(n log n)		mergesort, quicksort, heapsort
Θ(n^2)	négyzetes	sorbarendezés „legrosszabb” esetben
Θ(n^3)	köbös	n*n mátrixszorzás
Θ(n^c)	polinomiális	
Θ(c^n)	exponenciális	TSP exakt megoldás dinamikus programozással
Θ(n!)	faktoriális	TSP exakt megoldás brute force algoritmussal



Nagyságrendek





Tulajdonságok

- $f_1 \in O(g_1)$ és $f_2 \in O(g_2) \Rightarrow f_1 f_2 \in O(g_1 g_2)$
- $fO(g) \subset O(fg)$

- $f_1 \in O(g_1)$ és $f_2 \in O(g_2) \Rightarrow f_1 + f_2 \in O(|g_1| + |g_2|)$
  - ha $f_1 \in O(g)$ és $f_2 \in O(g) \Rightarrow f_1 + f_2 \in O(g)$

- $O(kg) = O(g)$

Heurisztikus algoritmusok

- A *heurisztikus algoritmusok* olyan technikák összefoglaló elnevezése, amelyek *a globális optimum, a teljesség vagy a pontosság* feláldozásával valamely nem exakt módszerrel próbálnak megoldást találni egy problémára.
- A vizsgált problémák esetében igaz, hogy a klasszikus, egyértelmű megoldásokat adó algoritmusok nem elég hatékonyak, vagy nem léteznek, így azok alkalmazása különböző megfontolásokból nem alkalmazható.

Kompromisszumok

- ***Optimalitás***: Ha több megoldás is létezik, garantálja-e a módszer, hogy a legjobbat érjük-e el? Egyáltalán mennyire fontos a legjobb megoldás megtalálása, vagy megfelel egy elég jó is?
- ***Teljesség***: *Ha több megoldás létezik, a heurisztika megtalálja-e mindet? Szükségünk van mindre?* A legtöbb heurisztika csak egyet talál meg.
- ***Pontosság***: Meg tudje-e adni a heurisztika az elért megoldás konfidencia intervallumát? Ez mennyire „nagy”?
- ***Végrehajtási idő***: Ez a legjobb ismert heurisztika a probléma megoldására? Néhány heurisztika nem jelentősen gyorsabb a klasszikus megoldásnál.

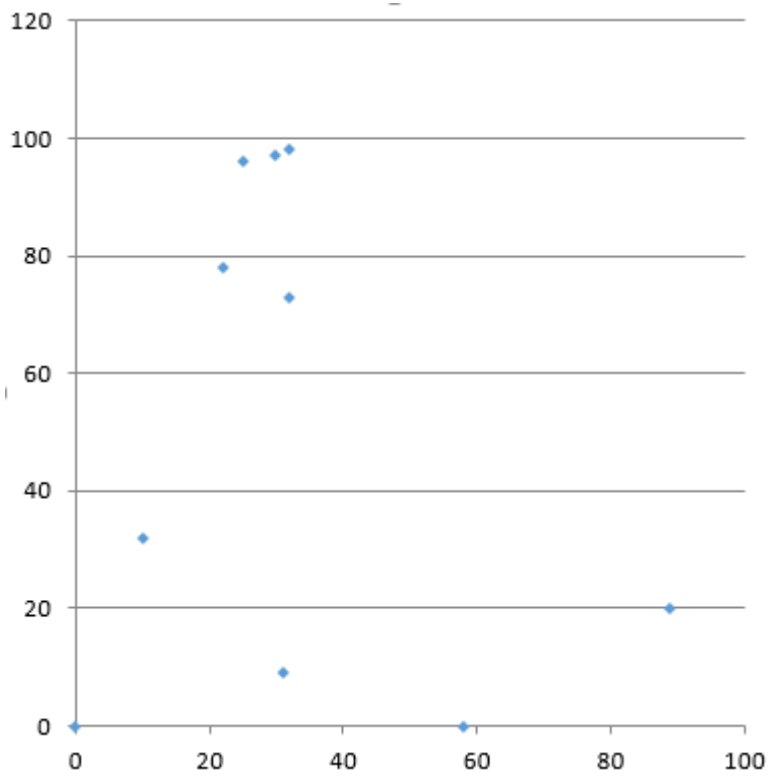
A heurisztika

- *Definíció*: Heurisztikának minősül minden olyan szabály, következtetés, értékelés, érv, melyről elmondható, hogy egy bizonyos fajta szituációban többnyire érvényes, illetve működik, de nem mindig.
- *Dinamikus heurisztikák*: ahol az a fő szerveződési elv, hogy adott szituációban milyen lépést célszerű választani
- *Statikus heurisztikák*: ahol a fő szerveződési elv az, hogy egy adott állást hogyan értékelünk.

Példa, Utazó ügynök probléma 1.

Az utazó ügynök problémája (Traveling Salesman Problem, TSP) egy kombinatorikus optimalizálási probléma. Kiváló példa a bonyolultság-elmélet által NP-nehéznek nevezett problémaosztályra.

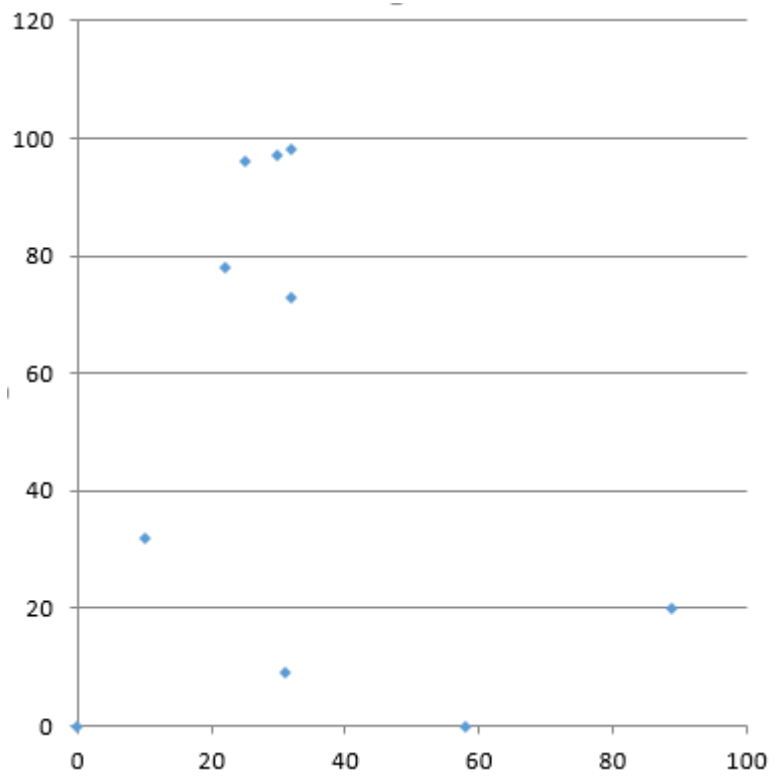
Adott *n* város, és az útiköltség bármely két város között, keressük a legolcsóbb utat egy adott városból indulva, amely minden várost pontosan egyszer érint, majd a kiindulási városba ér vissza.



Példa, Utazó ügynök probléma 1.

Az utazó ügynök probléma megoldása a városok valamely (gyűrűs) permutációja.

Az utak száma így: $\frac{(n-1)!}{2}$ (abban az esetben, ha két város közti távolság a két irányban megegyezik, és mivel gyűrűt alkot az út)



Példa, Utazó ügynök probléma 2.

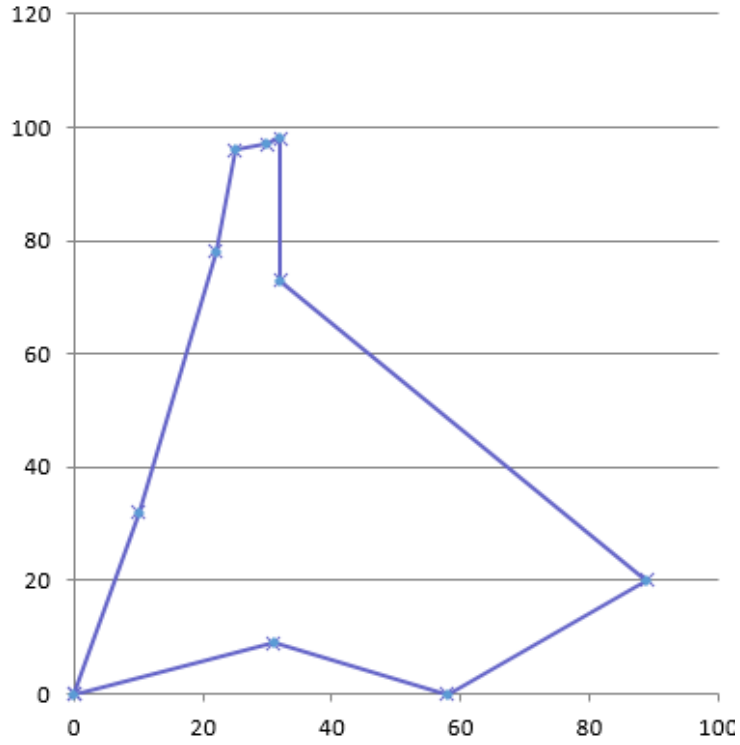
Az utazó ügynök p=$[a_1, a_2, \dots a_n, a_{(n+1)}(a_1)]$ útvonalához tartozó útvonal jósága

$f(p) = \sum_{i=1}^{n} D(a_i, a_{i+1})$

Kézenfekvő megoldás az összes útvonal megvizsgálása (brute force)

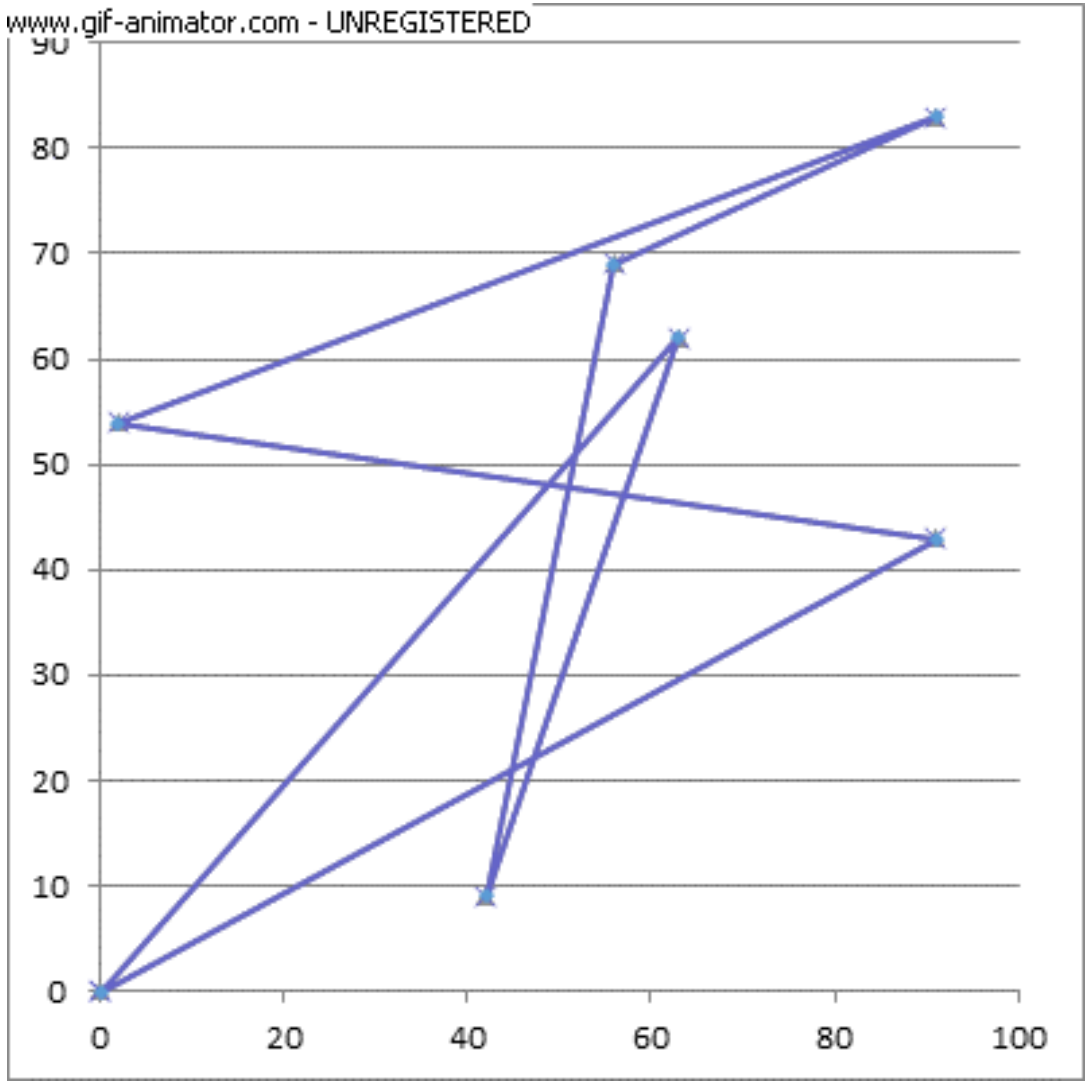
Ebben az esetben azonban a lehetséges permutációk növekedésével

(n>25 esetén a világ összes számítógépe nem találja meg a megoldást egy év alatt)



n	$\frac{(n-1)!}{2}$
10	181440
20	6,08E+16

Brute Force Példa



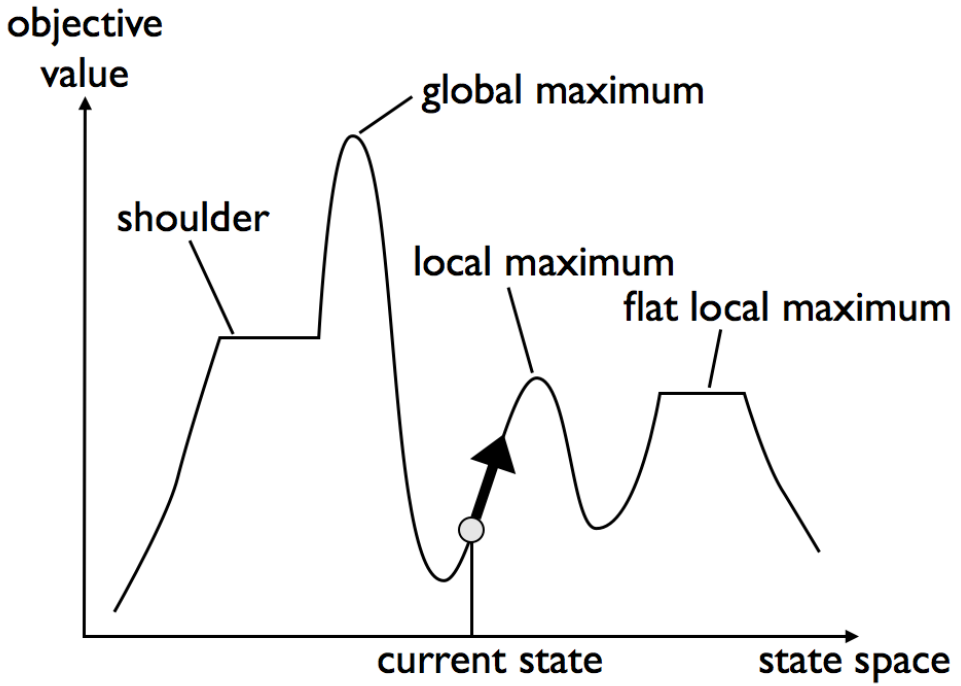
Hill Climbing, hegymászó algoritmus 1.

Lokális keresési módszer, amely egy kiinduló megoldást feltételezve, annak valamely elemét módosítva keres jobb megoldást. Amennyiben az új megoldás jobb, az lesz az új megoldás

Viszonylagos egyszerűsége miatt népszerű „első választás” lehet

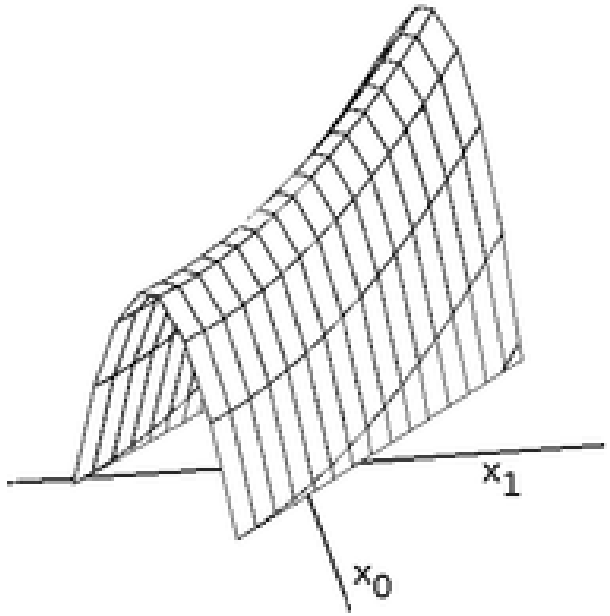
Csak lokálisan ad optimális megoldást

Többszöri indítással az eredmény javítható



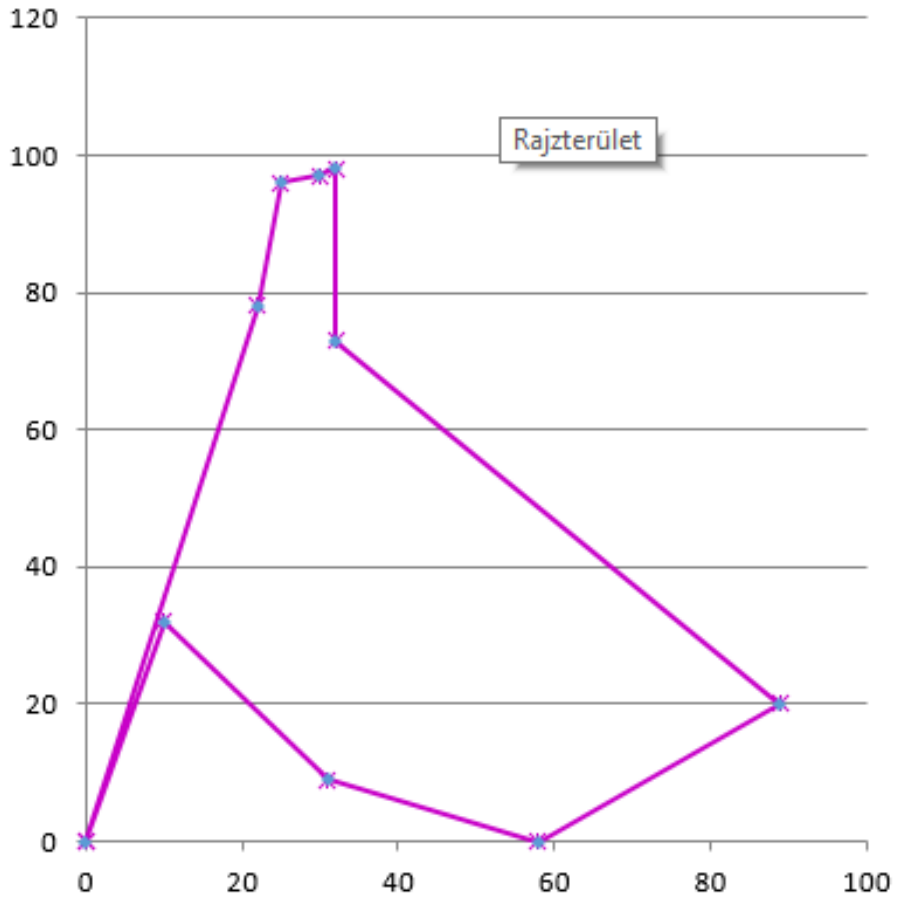
Hill Climbing, hegymászó algoritmus 2.

- (Egyszerű) Simple hill climbing: Minden lehetséges szomszéd kiértékelésre kerül, és a legjobb kerül kiválasztásra
- Stochastic Hill Climbing: Véletlen lépés kiválasztásával halad felfelé
- A platók (fennsíkok) és a nyergek nehéz döntés elé állítják az algoritmust
- (Lásd még: gradient search)

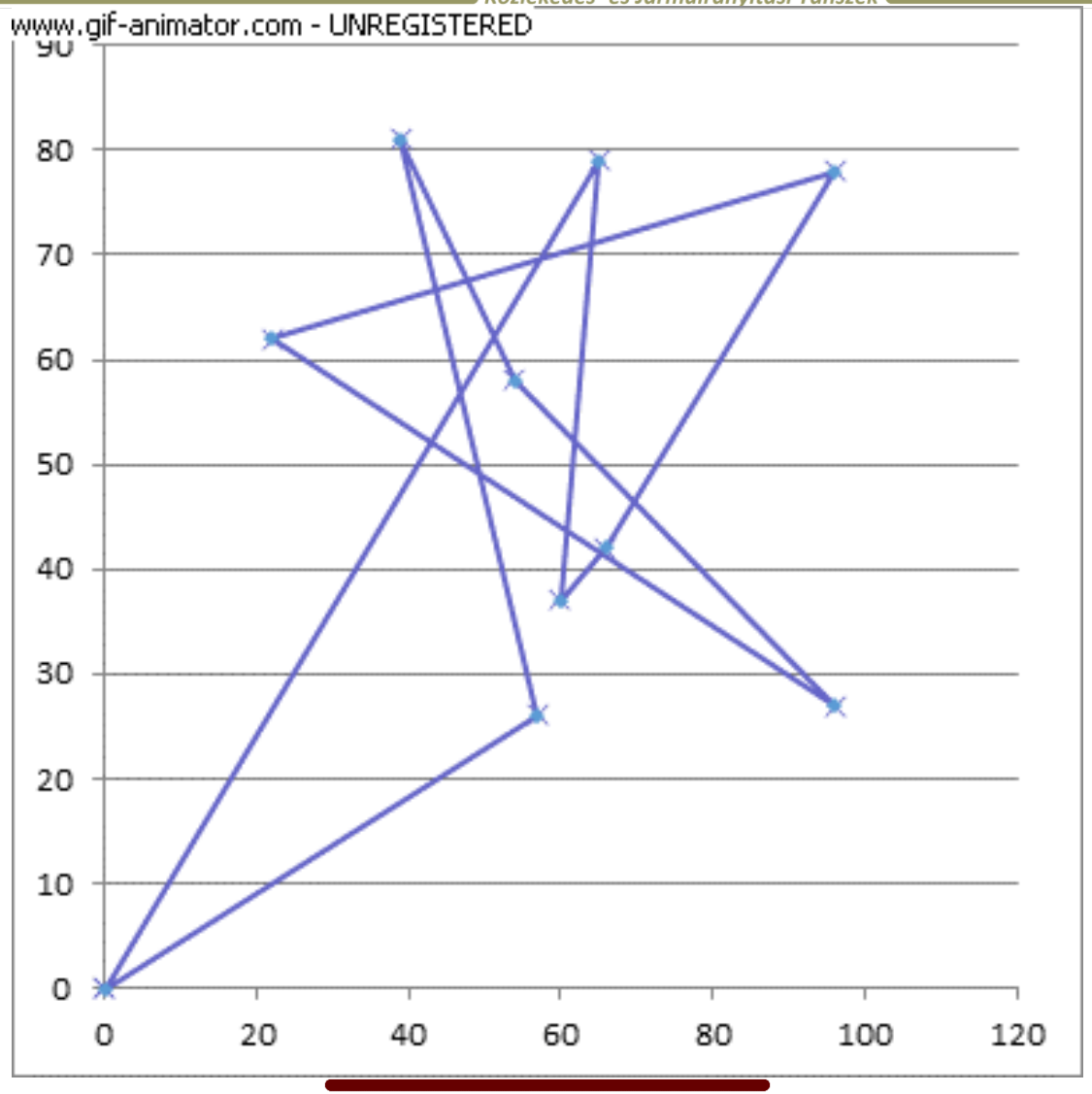


Sztochasztikus Hegymászó TSP példa

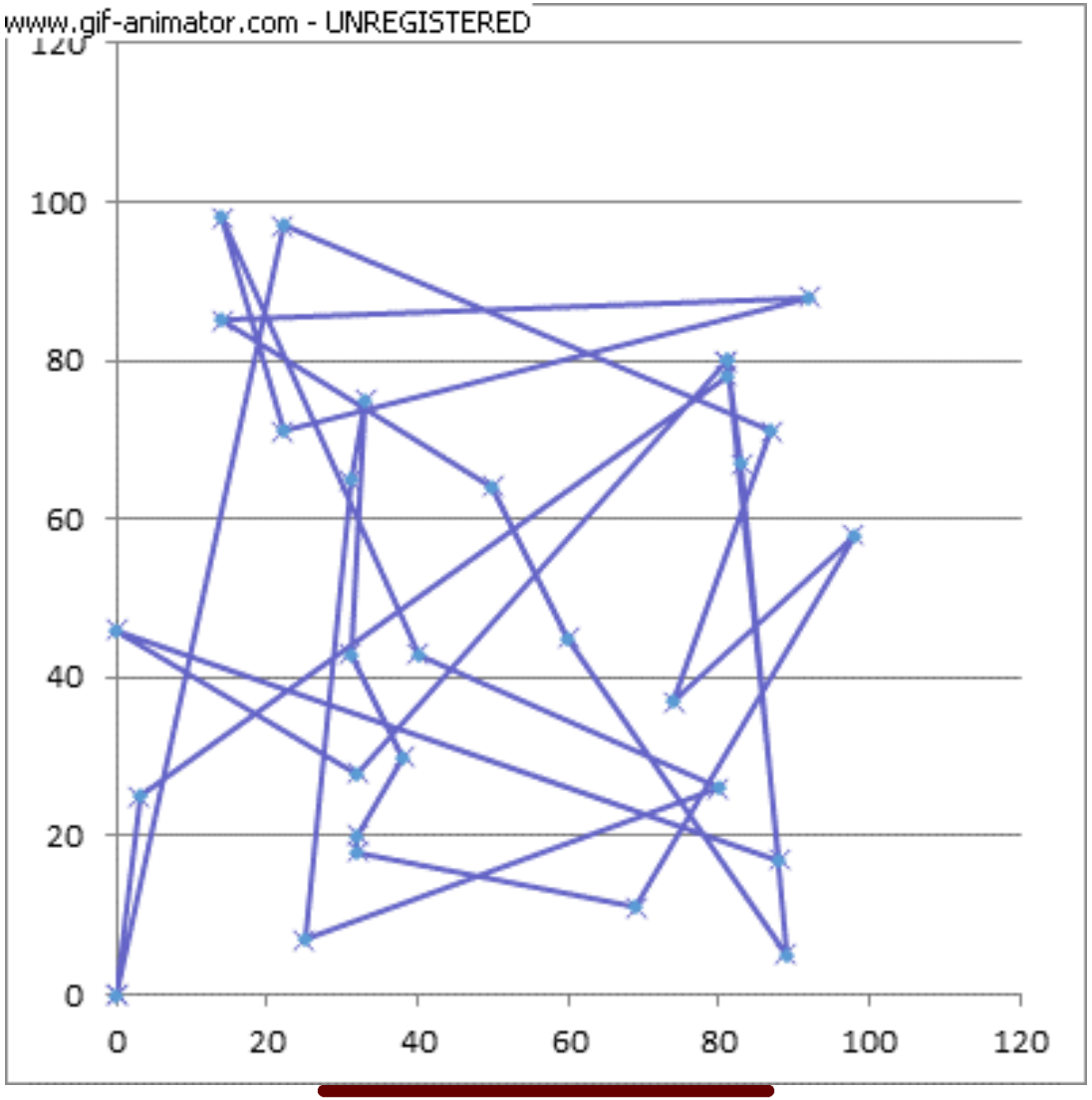
1. Keresünk egy véletlen kiinduló permutációt: p
2. Két véletlen elemet felcserélünk, így előáll p^*
3. Ha $f(p) > f(p^*)$, akkor $p = p^*$
4. Az algoritmus előre meghatározott lépésszám után véget ér
5. Az 1-4 lépéseket előre meghatározott számban végrehajtjuk



Random descent Példa 1



Random descent Példa 2



Mohó algoritusok (Greedy algorithms)

- A mohó algoritmus helyi optimumok megvalósításával próbálja megtalálni a globális optimumot. Lépései:

1. Kiinduló halmazból veszi a jelölteket, amelyekkel felállítja a megoldáshalmazt
2. Kiválasztás, amely a legjobb jelöltet választja ki
3. Megvalósíthatóság vizsgálat, amely eldönti, hogy egy jelölt alkalmas-e a megoldásra
4. Célfüggvény, a megoldásnak, részmegoldás jóságát dönti el
5. Megoldásfüggvény, amely jelzi, ha megtaláltuk a teljes megoldást.

Mohó algoritmus, címletek kiválasztása

- Adott egy pénzösszeg n, és címletek a={10,5,2,1}, határozzuk meg a minimális elemszámú érmesorozatot (s), amellyel n kifizethető:
  1. Keressük meg azt a legnagyobb pénzérmét, amelyre igaz, hogy a fennmaradó összegnél kisebb ($a_i < n - \sum s$)
  2. Vegyük fel a_i-t s-be.
  3. Ugorjunk vissza az 1. lépéshez, ha $n > \sum s$

  Példa: n=19 ; s=[10,5,2,1,1]
- A fenti példa (és a jelenleg használt címleteink) minden esetben globális optimumot ad.
- Abban az esetben, ha a={10,7,1} és n=15, az algoritmus s=[10,1,1,1,1,1] eredményt ad az optimális s=[7,7,1] helyet.

TSP mohó algoritmusok 1.

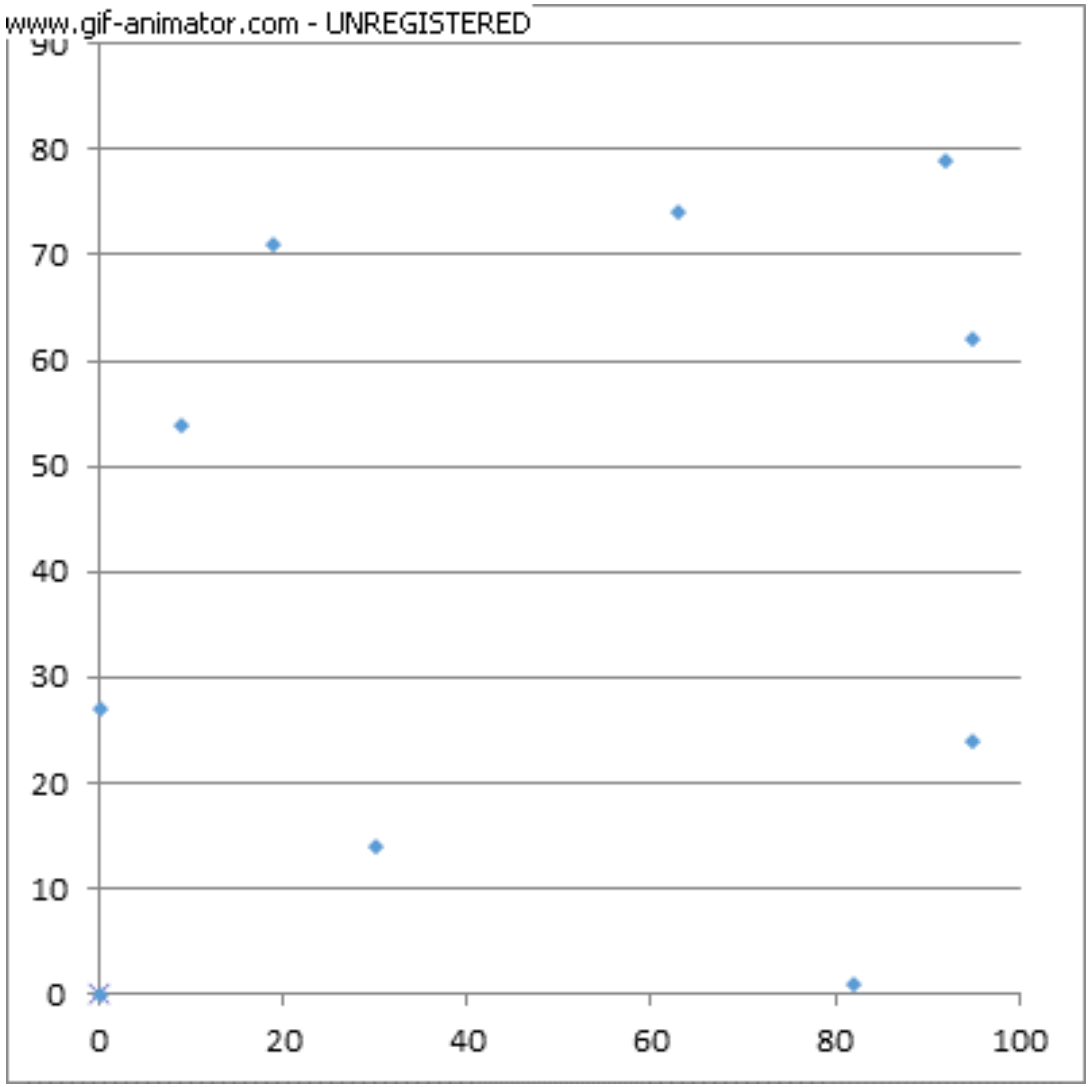
- Nearest Neighbor Algorithm *(Rosenkrantz, Stearns, Lewis, 1974)*

1. Kiinduló „útvonal” egy elemmel
2. Keressük meg azt az r pontot amely a legközelebb esik az útvonalunk végéhez, és vegyük fel
3. Vissza 1., amíg nem érintettünk minden pontot
4. Zárjuk be a kört, összekötve az utolsó elemet az elsővel

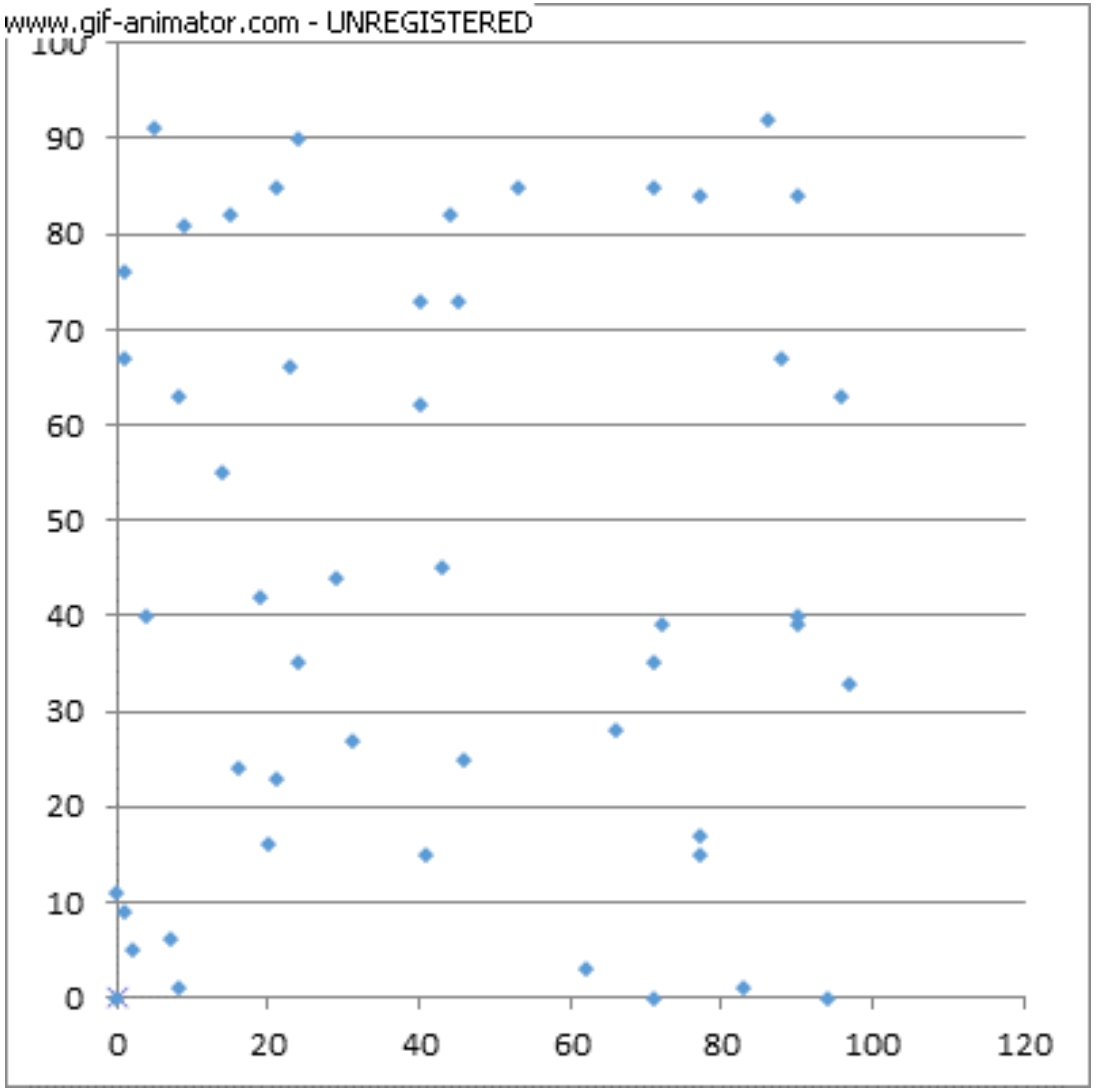
Számításigény: $O(n^2)$,

worst case: $\frac{p_{alg}}{p_{opt}} \leq \frac{1}{2}\lfloor \log_2(n) \rfloor + \frac{1}{2} \quad \lfloor \quad \rfloor - fix\ function$

Mohó példa 1


www.gif-animator.com - UNREGISTERED
90
80
70
60
50
40
30
20
10
0
0
20
40
60
80
100

Mohó példa 2


www.gif-animator.com - UNREGISTERED
100
90
80
70
60
50
40
30
20
10
0
0
20
40
60
80
100
120

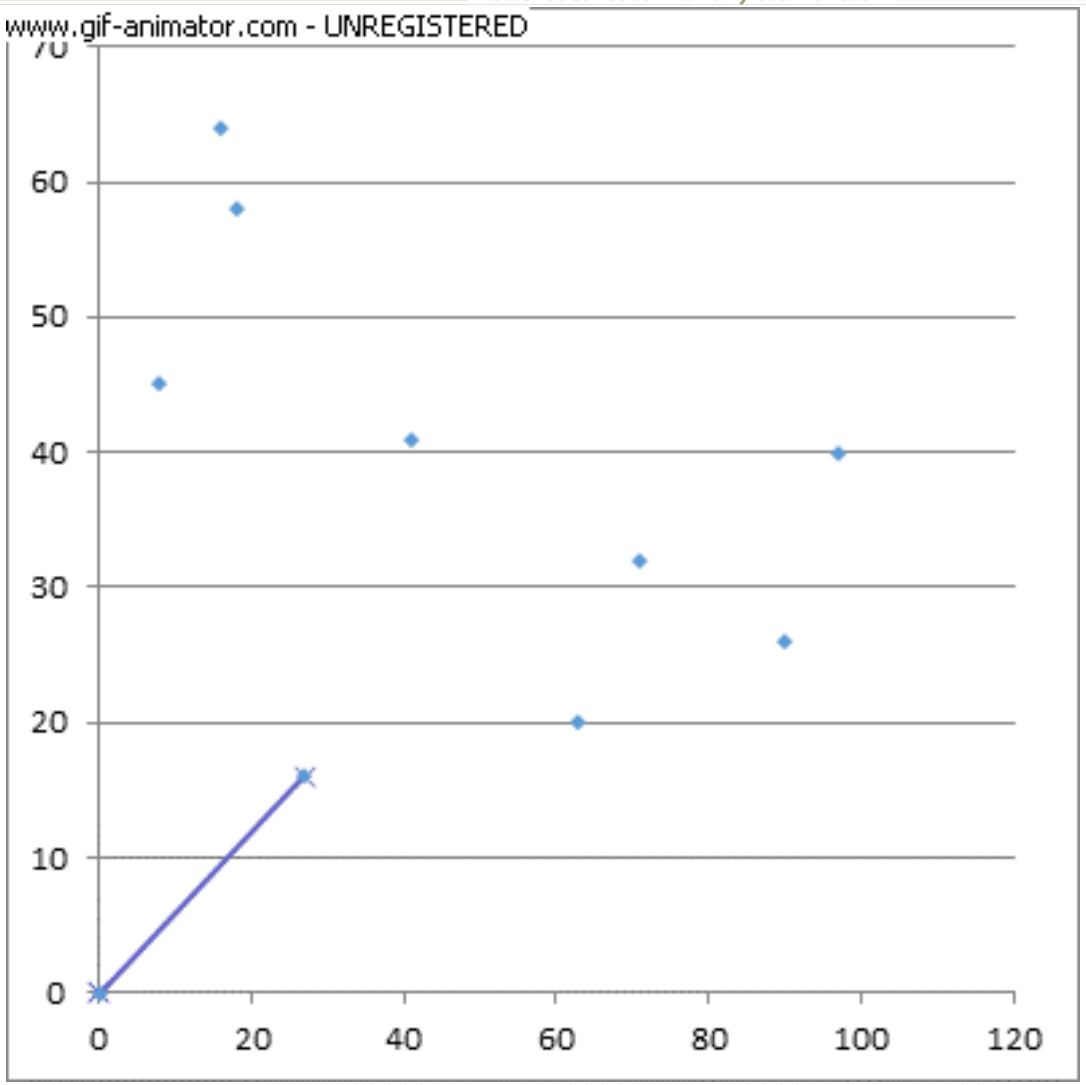
TSP mohó algoritmusok 2.

- Nearest Insertion *(Rosenkrantz, Stearns, Lewis, 1974)*

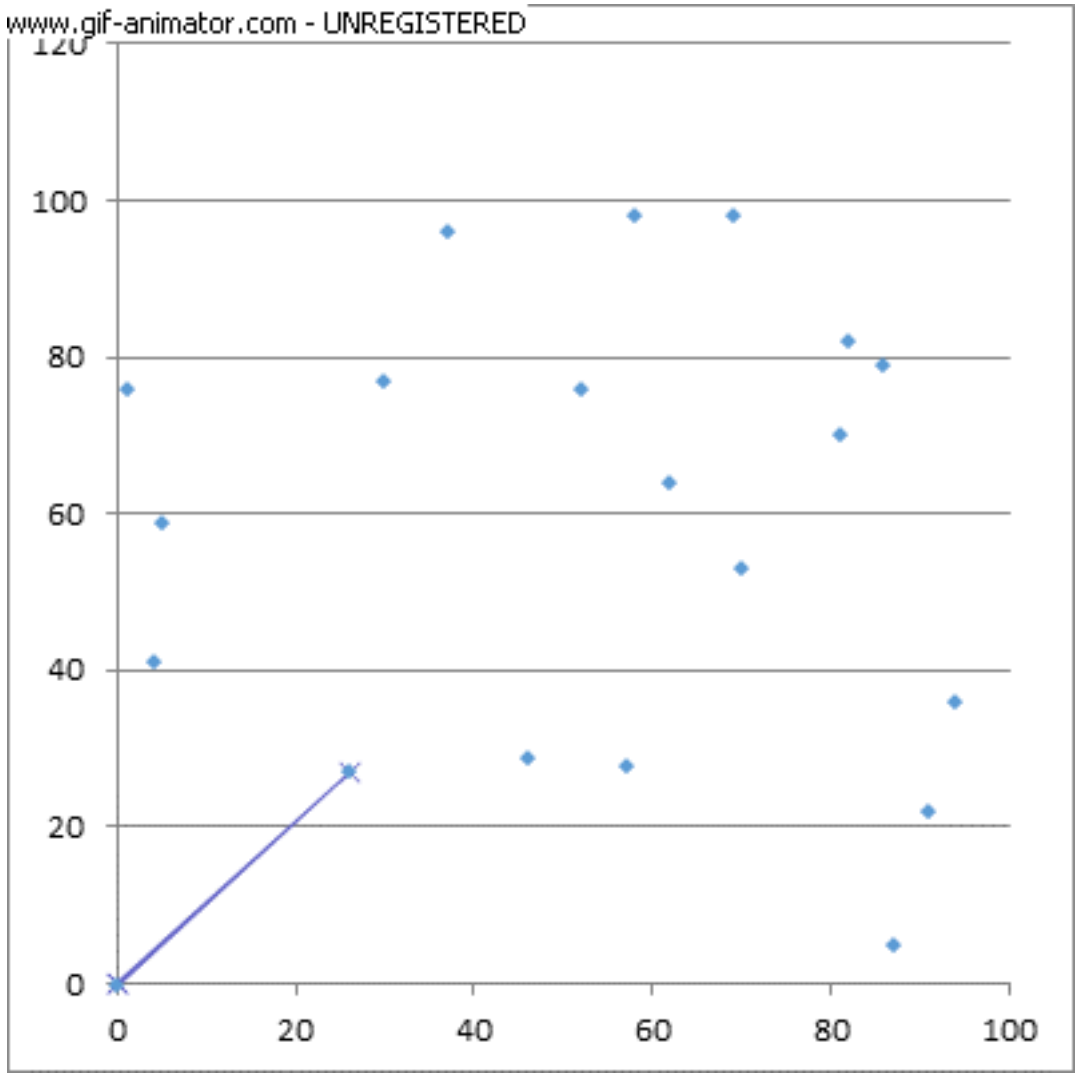
1. Kiinduló „útvonal" egy elemmel (i)
2. Keressük meg azt az r pontot, ahol c_{ir} minimális, legyen a kezdő körutunk i-r-i
3. (Kiválasztás) Keressük meg r pontot, amely nem szerepel már az útvonalban, és bármely útvonalponthoz a legközelebb esik
4. (Beillesztés) Keressük meg azt az (i,j) élet az útvonalon, ahol $c_{ir} + c_{rj} - c_{ij}$ minimális, illesszük be ide a pontot
5. Vissza 3., amíg nem érintettünk minden pontot

Számításigény:$O(n^2)$, worst case: $\frac{p_{alg}}{p_{opt}} \leq 2$

Nearest insertion Példa 1



Nearest insertion Példa 2



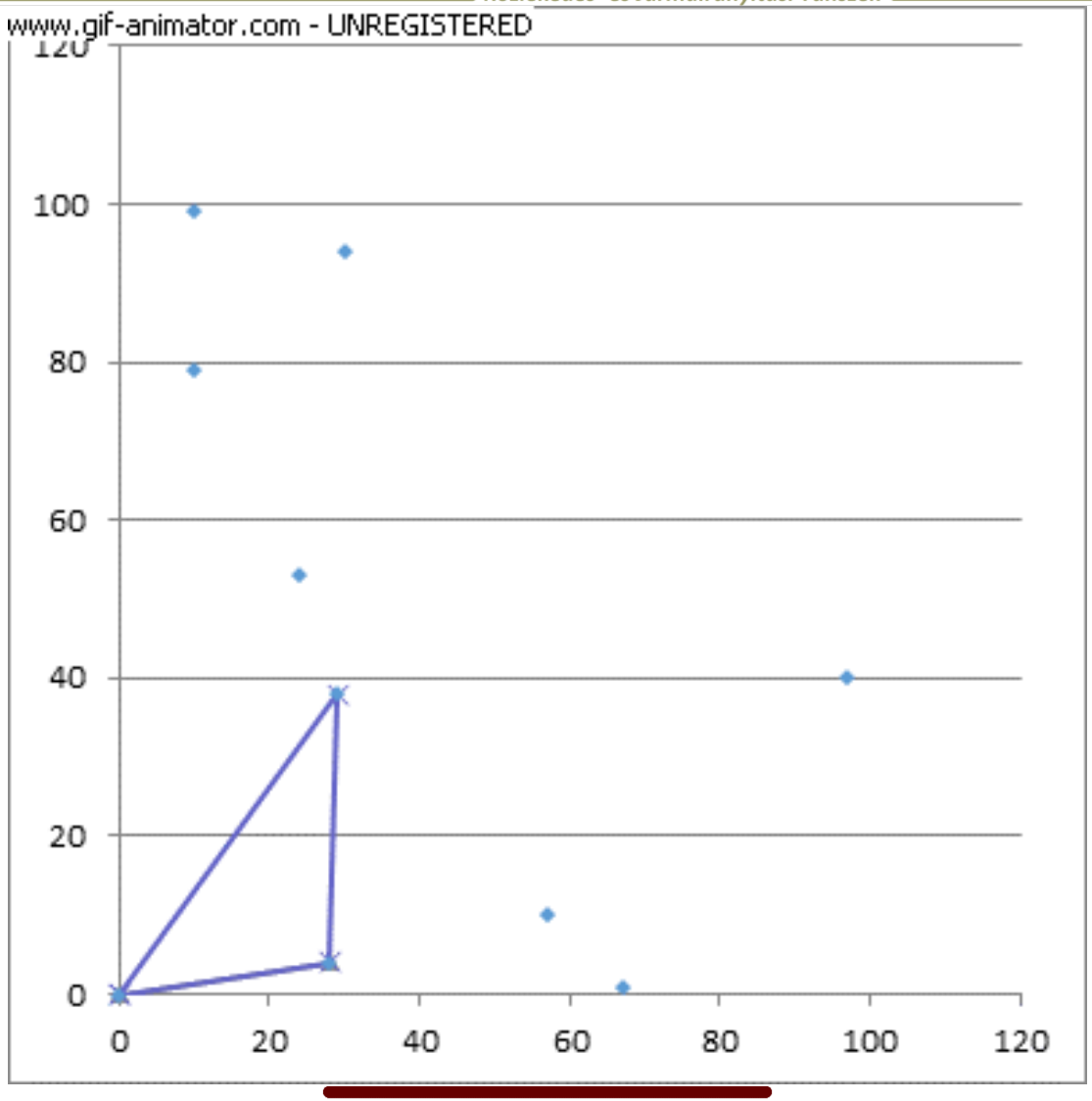
TSP mohó algoritmusok 3.

- Cheapest Insertion *(Rosenkrantz, Stearns, Lewis, 1974)*

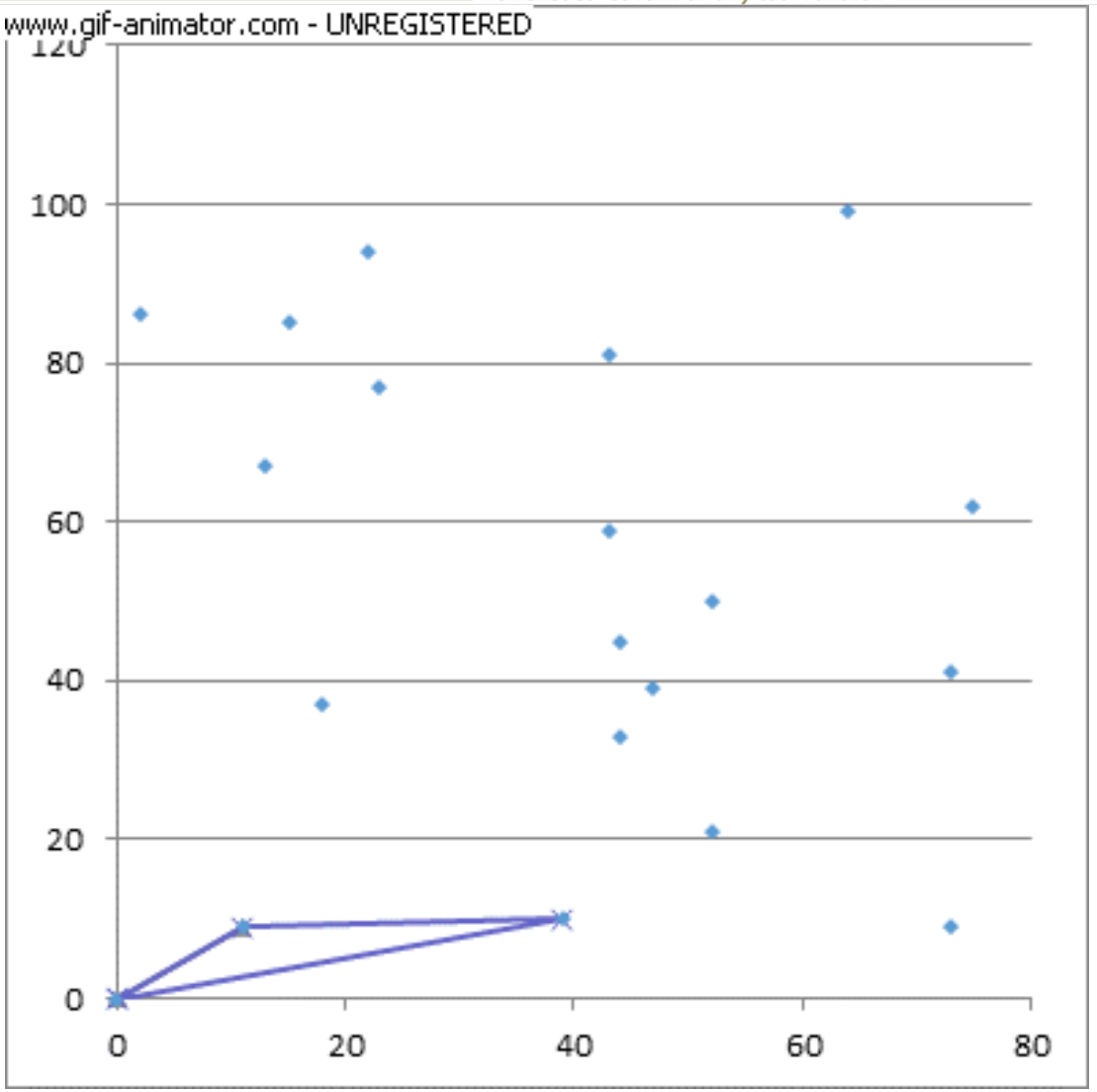
1. Kiinduló „útvonal” egy elemmel (i)
2. Keressük meg azt az r pontot, ahol c_{ir} minimális, legyen a kezdő körutunk i-r-i
3. Keressünk meg azt az (i,j) élet az útvonalon és r pontot (nem az útvonalon), ahol $c_{ir} + c_{rj} - c_{ij}$ minimális, illesszük be ide a pontot
4. Vissza 3., amíg nem érintettünk minden pontot

Számításigény:$O(n^2 \log_2(n))$, worst case: $\frac{p_{alg}}{p_{opt}} \leq 2$

Cheapest Insertion Példa 1



Cheapest Insertion Példa 2



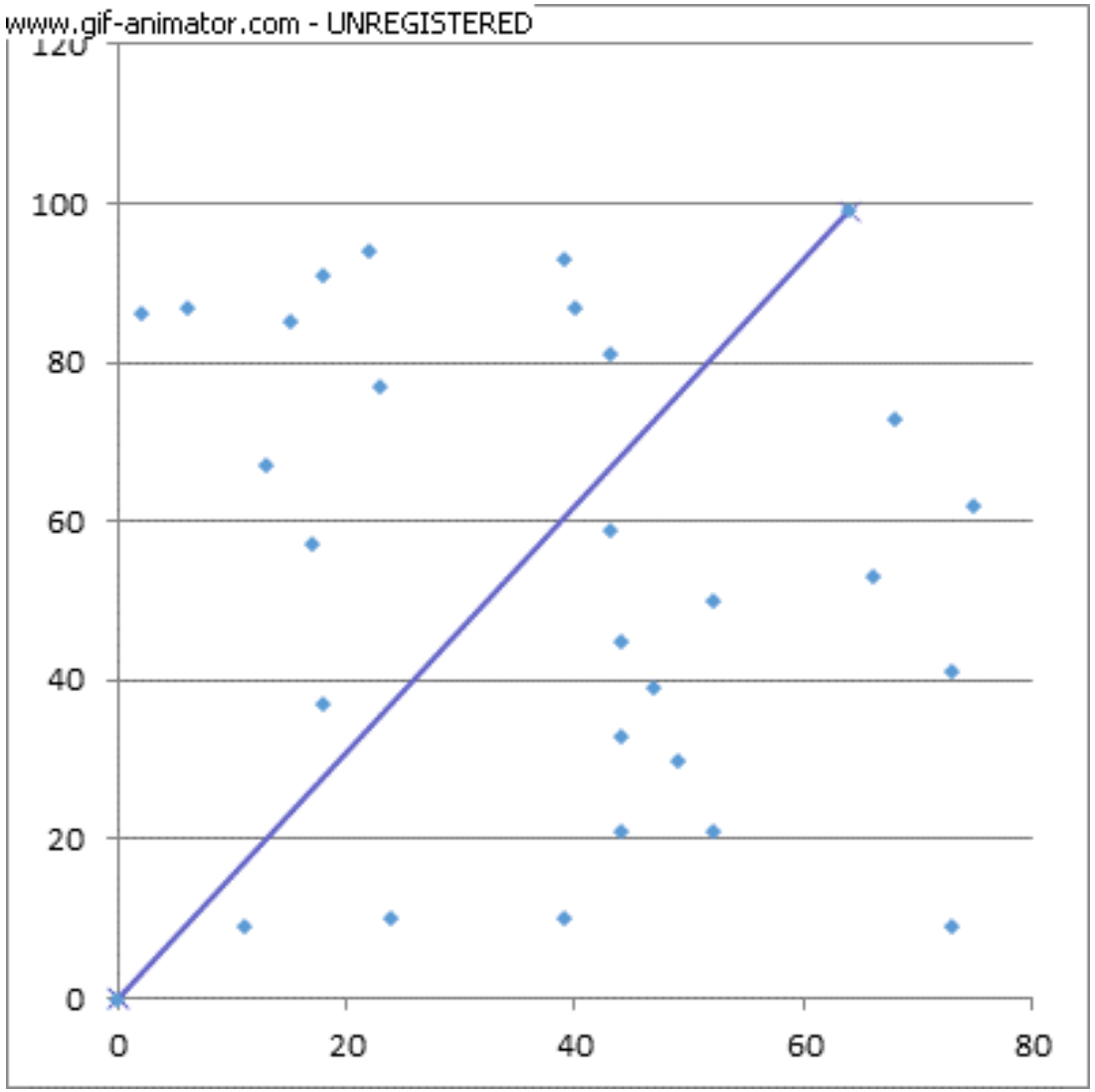
TSP Farthest insertion

- **„Ellentett" Mohó**

1. Keressük meg i,r párt, ahol ahol c_{ir} maximális, legyen a kezdő körutunk i-r-i
2. Keressük meg azt az r pontot amelyik nincs az útvonalon, és a legmesszebb van az útvonal bármelyik pontjától
3. Keressünk meg azt az (i,j) élet útvonalon, ahol $c_{ir} + c_{rj} - c_{ij}$ minimális, illesszük be ide a pontot
4. Vissza 2., amíg nem érintettünk minden pontot

Számításigény:$O(n^2)$

Farthest Insertion Példa



Szimulált hűtés, Simulated Annealing (SA)

- A heurisztikus algoritmusok sok esetben megrekedhetnek egy lokális minimumban, ahonnan képtelenek kitörni.
- A szimulált hűtés technikailag egy metaheurisztika, azaz egy magasabb szintű eljárás, amely kiszolgálja a keresőalgoritmust, és segíti kibillenteni egy beragadt állapotból.
- Az SA a fémek hűtésének példájából veszi az inspirációt, lényege, hogy a megfelelően kontrolált hűtési folyamat során optimális kristályméretet tudunk elérni.

Az SA alapelve

- A lokális minimumban maradás kiküszöbölésére az alkalmazott heurisztikát nem csak az optimum irányába engedjük eltolódni, hanem bizonyos valószínűséggel azzal ellentétes irányba is.
- Az SA ennek megfelelően az algoritmus futása közben egy (általában) folyamatosan csökkenő hőmérséklet értéket kezel, amely meghatározza az új jelölt elfogadásának valószínűségét.

SA Elfogadási kritérium

- Az SA egy elért **s** állapot, és az ezzel szomszédos, ezt követő **s'** állapot közül kell, hogy kiválassza a következőt **T** hőmérséklet függvényében. Az **s** és **s'** állapotok jósága **e** és **e'**.
- Ez a valószínűségi függvény ekkor P(e,e',T) formájú
- A legtöbbször, az elfogadás valószínűsége **1**, ha **e'<e**
- Kritérium, hogy amennyiben T tart nullához, akkor a P függvény is nullához tartson. T=0 esetén mohó algoritmust, vagy egyszerű hegymászó algoritmust kapunk

SA hűtés ütemezése

- Bizonyított, hogy az SA eléri a globális optimumot megfelelő hosszú hűtési hosszal, de ez praktikusan nem hasznos, mivel ez a hossz sok esetben meghaladhatja a brute force algoritmus számításigényét.
- A klasszikus elfogadási kritérium:
- $$P(e, e', T) = \begin{cases} 1 & , ha \;\; e' < e \\ e^{\frac{-(e'-e)}{T}} & , egyébként \end{cases}$$

TSP, szomszédos jelöltek keresése

- A TSP egy permutációs probléma, két jelölt szomszédos lehet, ha:
  - Egy pont átkerül egy másik helyre
  - Két szomszédos pont sorrendje felcserélődik
  - Két tetszőleges pont helyet cserél (Erre mutat példát a következő demo)

Szimulált hűtés példa