Komplex Tervezés:
Éttermi kiszállítás szimulációja és optimalizációja

Reisz Ákos
FZ3S16

Feladat

Az éttermi rendelések kiszállításánál megjelenő optimalizálási problémák szimulációja és vizsgálata. Az optimalizálás célja, hogy minél több megrendelést, minél rövidebb idő alatt, minél kisebb költséggel lehessen teljesíteni. A szimuláció és az optimalizálás Python programozási nyelv segítségével készül.

Az éttermi rendelések kiszállításánál megjelenő optimalizálási problémák egészen visszavezethetők az utazó ügynök problémájához. Az utazó ügynök problémája (Travelling salesman problem, TSP) egy kombinatorikus optimalizálási probléma. Kiváló példa a bonyolultság-elmélet által NP-nehéznek nevezett problémaosztályra. Az utazó ügynök problémájához kapcsolódó matematikai feladatokkal először Sir William Rowan Hamilton és Thomas Penyngton Kirkman foglalkoztak az 1800-as években. Adott n darab pont és páronként az egymástól való távolságuk. Egy ügynök mindegyik pontot meg akarja látogatni (tetszőleges pontból kiindulva), de mindegyiket csak egyszer. A feladat az, hogy határozzunk meg egy olyan körutat, amely minimális hosszúságú legyen. Ha feltesszük, hogy bizonyos pontokból nem lehet közvetlenül eljutni egyes pontokba, míg a többi pontba egységnyi költséggel lehet eljutni és az ügynöknek minden pontot meg kell látogatnia.

Összes lehetséges út:

$$\frac{(n-1)!}{2}$$

Ezek közül kell választanunk, ez ugyanis a Hamilton-körök száma az n pontú teljes gráfban.

A képlet csak n>2 esetén működik.

Ezen importok szükségesek a szimuláció futtatásához

```
import random, numpy, math, copy, matplotlib.pyplot as plt, pandas as pd
from scipy.spatial import distance_ matrix
```

A cél az, hogy listát készítsünk a pontokról, amelyek mindegyike két koordinátát tartalmaz (x, y), amelyek 0 és 100 közötti véletlen egész számokként kerülnek kiválasztásra. Jelen esetben 10 ilyen pont lesz.

```
Points = [random.sample(range(100), 2) for x in range(10)];
```

A pontok közötti távolságok kimutatását így oldottam meg.

```
data = Points
points = ['1', '2', '3', '4', '5', '6', '7', '8', '9', '10']
df = pd.DataFrame(data, columns=['xcord', 'ycord'], index=points)
pd.DataFrame(distance_ matrix(df.values, df.values), index=df.index, columns=df.index)
```

Ezt a kimenetett adta válaszul.

	1	2	3	4	5	6	7	8	9	10
1	0.000000	24.207437	13.892444	11.704700	52.000000	51.088159	27.802878	31.827661	60.926185	56.080300
2	24.207437	0.000000	37.483330	31.953091	75.591005	72.277244	36.674242	55.973208	84.534017	80.280757
3	13.892444	37.483330	0.000000	16.492423	41.773197	44.911023	35.440090	18.867962	50.447993	43.266615
4	11.704700	31.953091	16.492423	0.000000	44.102154	40.607881	18.973666	28.635642	53.000000	51.224994
5	52.000000	75.591005	41.773197	44.102154	0.000000	17.262677	52.201533	25.079872	8.944272	16.763055
6	51.088159	72.277244	44.911023	40.607881	17.262677	0.000000	42.201896	33.837849	21.587033	33.955854
7	27.802878	36.674242	35.440090	18.973666	52.201533	42.201896	0.000000	44.407207	60.307545	63.560994
8	31.827661	55.973208	18.867962	28.635642	25.079872	33.837849	44.407207	0.000000	33.060551	24.413111
9	60.926185	84.534017	50.447993	53.000000	8.944272	21.587033	60.307545	33.060551	0.000000	17.691806
10	56.080300	80.280757	43.266615	51.224994	16.763055	33.955854	63.560994	24.413111	17.691806	0.000000

A travel egy 10 számból álló lista, amely a pontok meglátogatására utal. Feltételezzük, hogy zárt hurokra van szükség, így az utolsó pont autómatikusan csatlakozik az elsőhöz.

```
travel = random.sample(range(10), 10);
```

Elindítunk egy ciklust az adott értékekkel

```
for tlp in numpy.logspace(0, 5, num = 100000)[::-1]:
```

Két pont véletlenszerű cseréjével új új utat képzünk. Úgy valósítom meg, hogy választok két számot az i-t és a j-t. Összeállítom a newTravel-t a régi travel másolásával az i indexig, majd összefűzöm a j-edik travelel-t és egészen folytatom addig, amíg a j nem éri el az i-edik pontot, majd befejezem a travel többi részét.

```
[i, j] = sorted(random.sample(range(10), 2));
newTravel = travel[:i] + travel[j:j + 1] + travel[i + 1:j] + travel[i:i + 1] + travel[j + 1:];
```

Ha az if értéke igaz akkor a travel megkapja a newTravel értékét, az előzöekben említett csere miatt ez már változott. Az elképzelés az, hogy minimalizálni szeretnénk a pontok közti távolságok költségének összegét. Ehhez a Gibb-s faktor-t használtam fel, aminek lényege, az új állapotba való átmenet valószínűsége. Csak az i-edik és j-edik pontok közötti távolságokat szükséges összegezni, mivel a többi távolság ugyanaz mint a travel-ben mint a newTravel-ben egyaránt. Ha a faktor > 1 akkor az új költség alacsonyabb, travel megkapja a newTravel értékét.

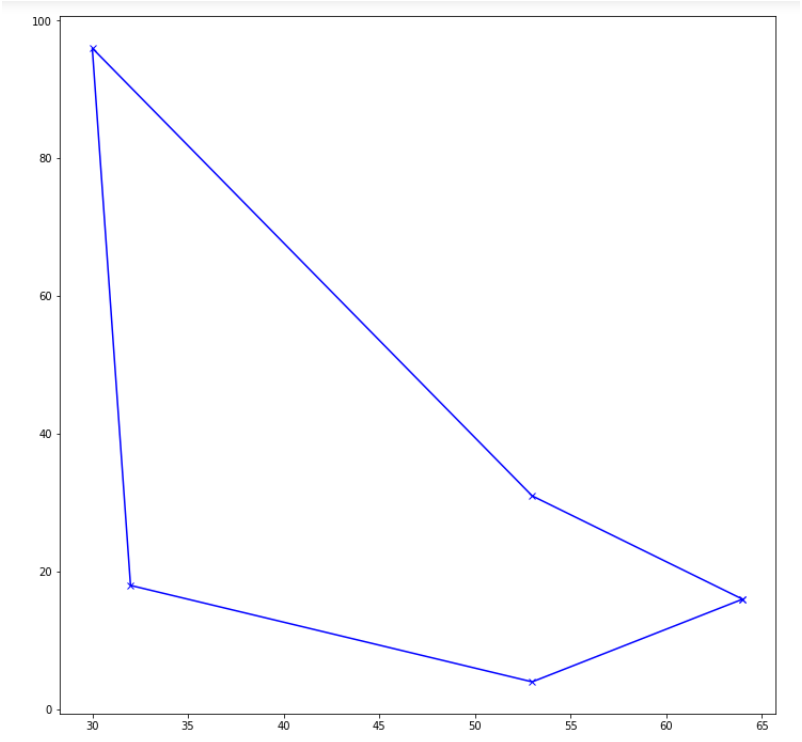
```
traveld = sum([math.sqrt(sum( [(points[travel[(k + 1) % 10]][d] - points[travel[k % 10]][d]) **
2 for d in [0, 1]])) for k in [j, j - 1, i, i - 1]])
newTraveld = sum([math.sqrt(sum([(points[newTravel[(k + 1) % 10]][d] - points[newTravel[k
% 10]][d]) ** 2 for d in [0, 1]])) for k in [j, j - 1, i, i - 1]])
if math.exp((traveld - newTraveld) / tlp) > random.random():
travel = copy.copy(newTravel);
```

Az algoritmus végeztével már csak meg kell jeleníteni a kívánt pontokat, ez kirajzol egy gráfot, amely optimális utat ad. Ehhez a pyplot libary-t használtam

```
plt.plot([points[travel[i % 10]][0] for i in range(11)], [points[travel[i % 10]][1] for i in
range(11)], 'xb-');
plt.show()
```

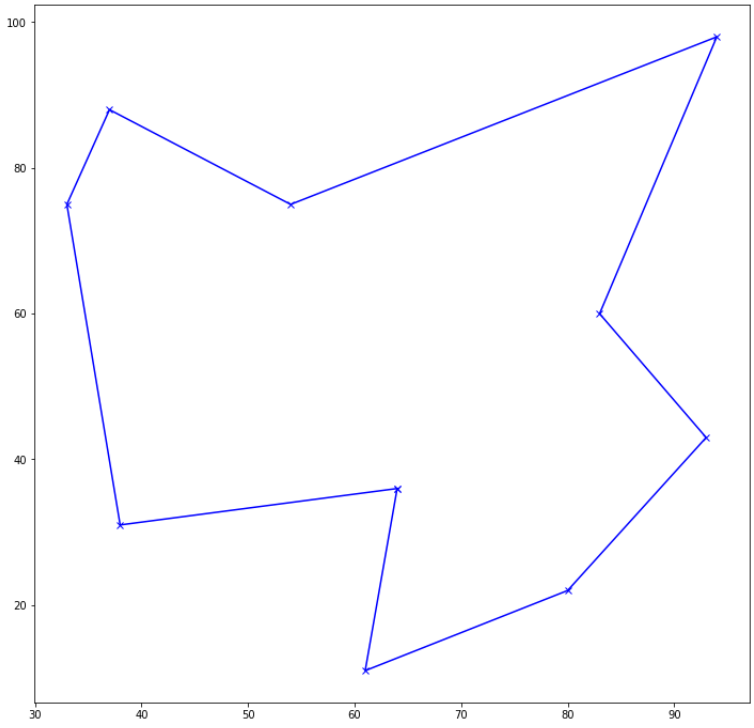
A következő gráfok megmutatják a kívánt utat városok száma szerint. Mivel véletlenszerű számok összehasonlításán alapszik az algoritmus ezáltal az összehasonlítások száma igen magas, viszont stagnál bizonyos értékek között.

5 város esetén



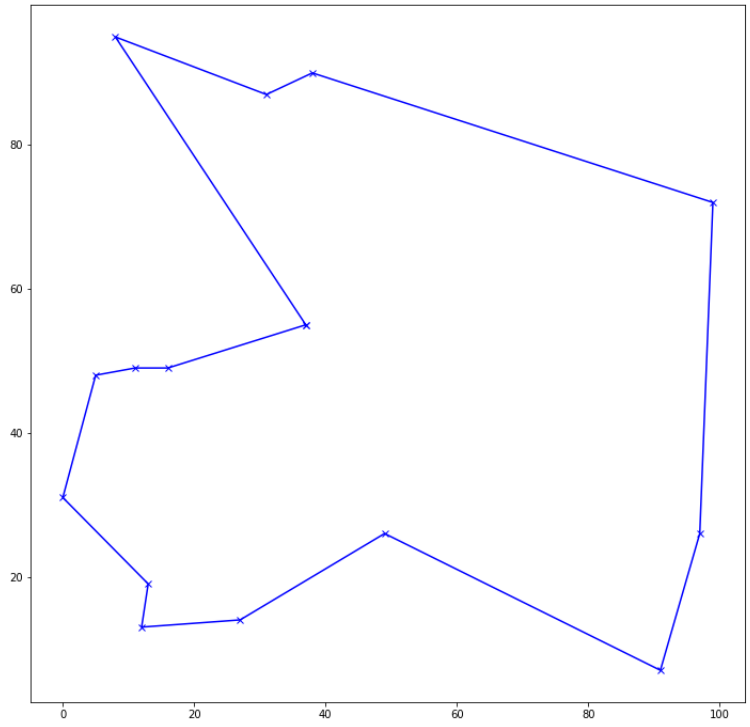
Összehasonlítások száma: 74 434

10 város esetén



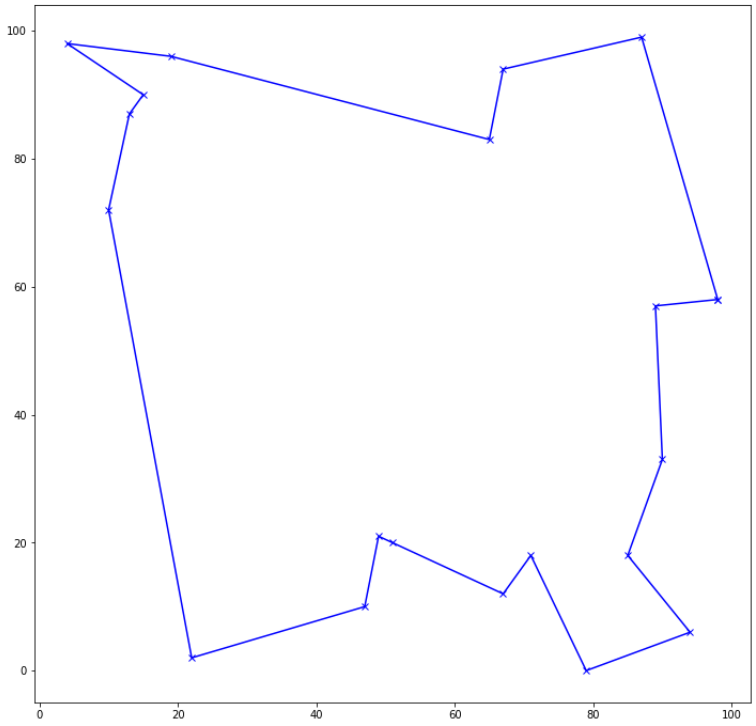
Összehasonlítások száma: 68 594

15 város esetén



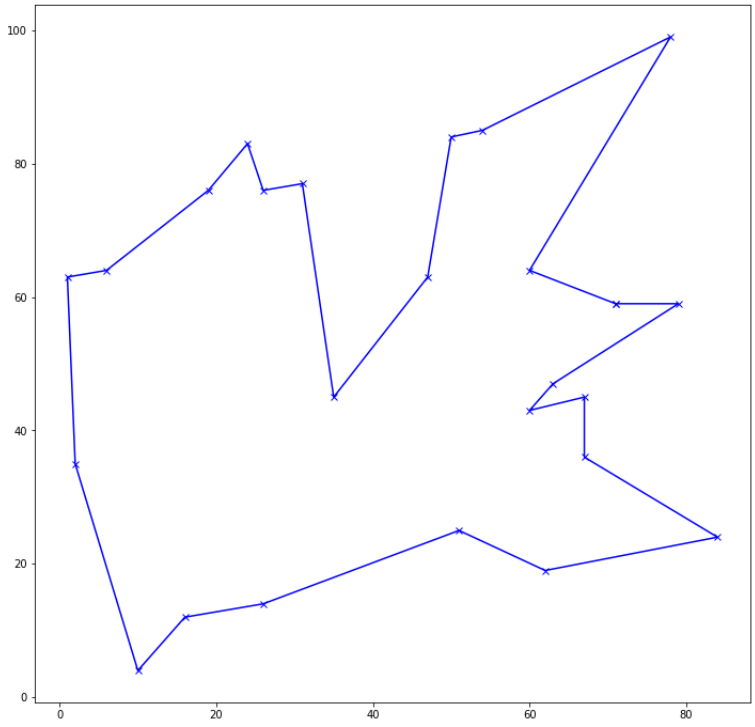
Összehasonlítások száma: 66 672

20 város esetén



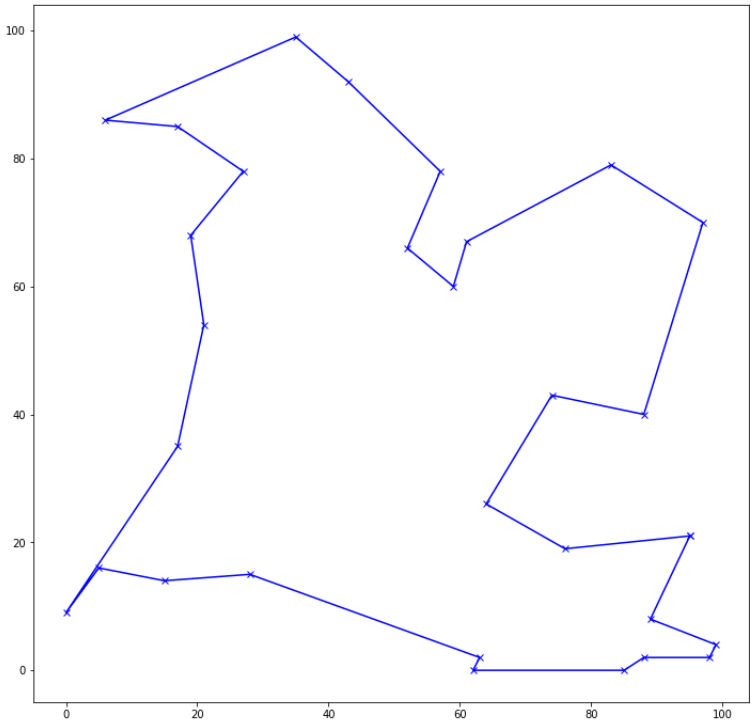
Összehasonlítások száma: 65 265

25 város esetén



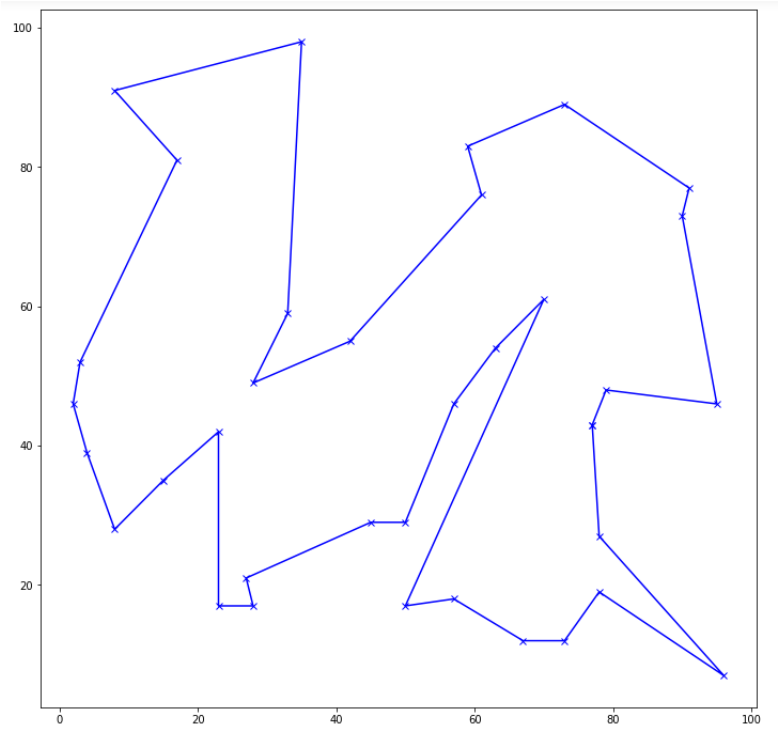
Összehasonlítások száma: 67 865

30 város esetén



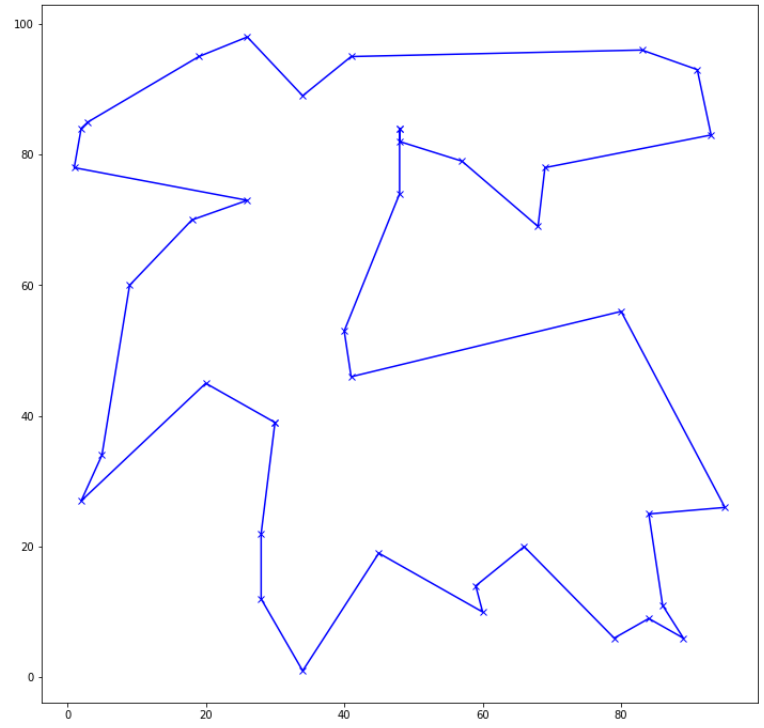
Összehasonlítások száma: 65 324

35 város esetén



Összehasonlítások száma: 67 230

40 város esetén



Összehasonlítások száma: 66 008

Több lefuttatott teszt után is megfgyelhető, hogy az összehasonlítások száma 65 ezer és 75 ezer között mözög. Egytől egyig a legoptimálisabb utat adták