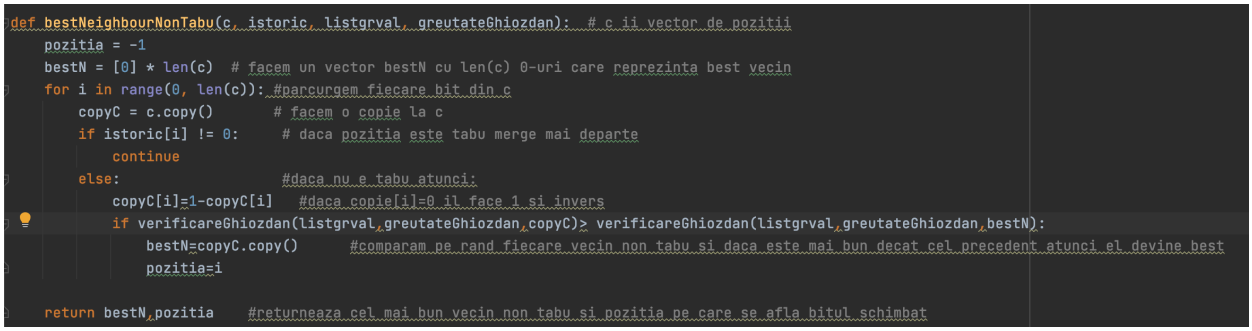
Tema2

Horeanga Bogdan gr.332

22 Martie 2021

Tabu Search

Figure 1

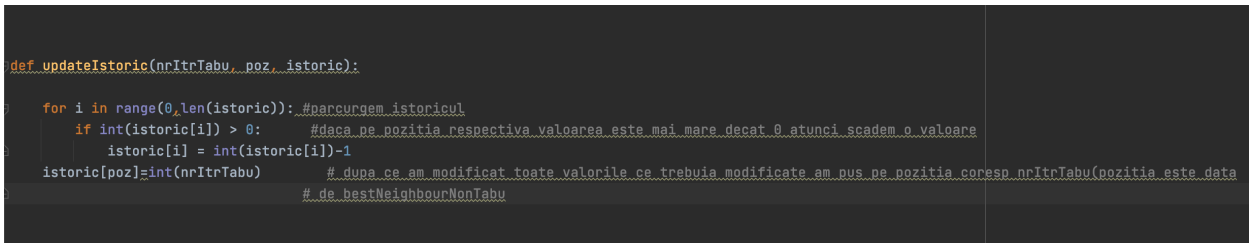


```
def bestNeighbourNonTabu(c, istoric, listgrval, greutateGhiozdan):  # c ii vector de pozitii
    pozitia = -1
    bestN = [0] * len(c)  # facem un vector bestN cu len(c) 0-uri care reprezinta best vecin
    for i in range(0, len(c)): #parcurgem fiecare bit din c
        copyC = c.copy()       # facem o copie la c
        if istoric[i] != 0:     # daca pozitia este tabu merge mai departe
            continue
        else:                   #daca nu e tabu atunci:
            copyC[i]=1-copyC[i]   #daca copie[i]=0 il face 1 si invers
            if verificareGhiozdan(listgrval,greutateGhiozdan,copyC)> verificareGhiozdan(listgrval,greutateGhiozdan,bestN):
                bestN=copyC.copy()      #comparam pe rand fiecare vecin non tabu si daca este mai bun decat cel precedent atunci el devine best
                pozitia=i

    return bestN,pozitia    #returneaza cel mai bun vecin non tabu si pozitia pe care se afla bitul schimbat
```

Algormitul care determina cel mai bun vecin non tabu primeste ca parametrii vectorul c(de pozitii), istoricul, lista cu greutatile si valorile si greutatea maxima ce incape in ghiozdan. Se initializeaza bestN cu un vector de len(c) 0-uri si pozitia=-1. Parcurge fiecare pozitie a vectorului c, daca pozitia asociata este tabu(adica se afla in istoric) atunci creste i-ul. Alt fel se transforma bitul din 0 in 1 si invers, se verifica daca e acest vecin este mai bun decat bestN(dar cum a fost 0, bestN se transforma pe rand cel mai bun vecin). Se returneaza cel mai bun vecin non tabu si pozitia pe care sa facut bit flip.

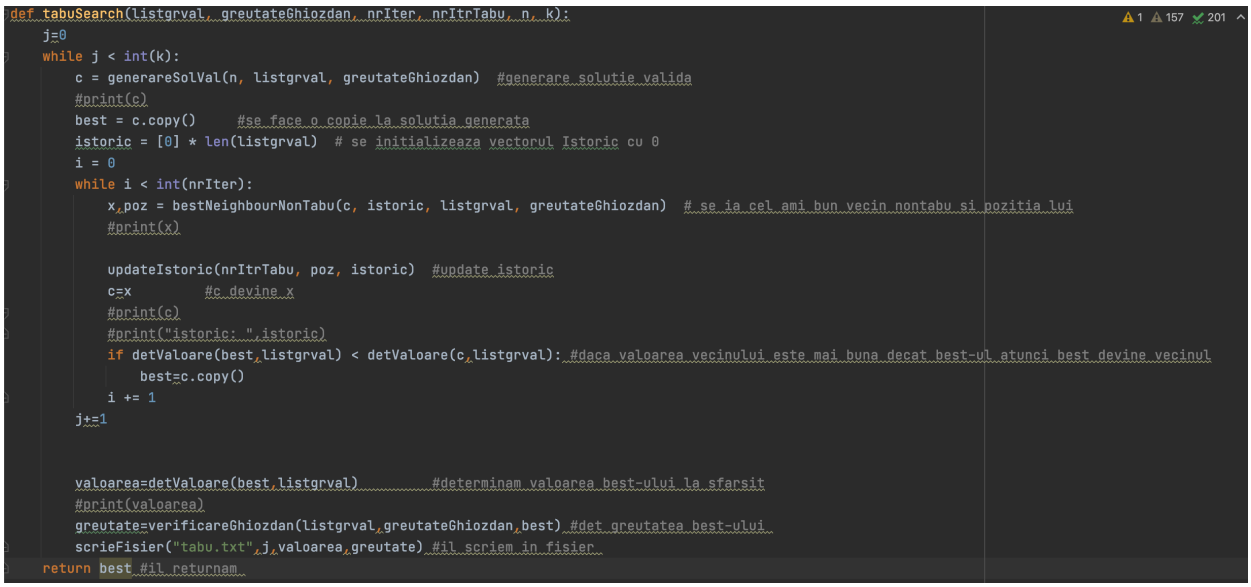
Figure 2



```
def updateIstoric(nrItrTabu, poz, istoric):

    for i in range(0,len(istoric)): #parcurgem istoricul
        if int(istoric[i]) > 0:      #daca pe pozitia respectiva valoarea este mai mare decat 0 atunci scadem o valoare
            istoric[i] = int(istoric[i])-1
    istoric[poz]=int(nrItrTabu)        # dupa ce am modificat toate valorile ce trebuia modificate am pus pe pozitia coresp nrItrTabu(pozitia este data
                                # de bestNeighbourNonTabu
```

Figure 3



```
def tabuSearch(listgrval, greutateGhiozdan, nrIter, nrItrTabu, n, k):
    j=0
    while j < int(k):
        c = generareSolVal(n, listgrval, greutateGhiozdan)  #generare solutie valida
        #print(c)
        best = c.copy()     #se face o copie la solutia generata
        istoric = [0] * len(listgrval)  # se initializeaza vectorul Istoric cu 0
        i = 0
        while i < int(nrIter):
            x,poz = bestNeighbourNonTabu(c, istoric, listgrval, greutateGhiozdan)  # se ia cel ami bun vecin nontabu si pozitia lui
            #print(x)

            updateIstoric(nrItrTabu, poz, istoric)  #update istoric
            c=x         #c devine x
            #print(c)
            #print("istoric: ",istoric)
            if detValoare(best,listgrval) < detValoare(c,listgrval): #daca valoarea vecinului este mai buna decat best-ul atunci best devine vecinul
                best=c.copy()
            i += 1
        j+=1


        valoarea=detValoare(best,listgrval)          #determinam valoarea best-ului la sfarsit
        #print(valoarea)
        greutate=verificareGhiozdan(listgrval,greutateGhiozdan,best) #det greutatea best-ului
        scrieFisier("tabu.txt",j,valoarea,greutate) #il scriem in fisier
    return best #il returnam
```

Simulated Annealing

Figure 4

```
def distEuclidiana(punctA, punctB):
    return int(sqrt(pow((punctB[0]-punctA[0]),2)+ pow((punctB[1]-punctA[1]),2)))


import random


def solinitialaTsp(n):
    randomlist = random.sample(range(1, int(n)+1), int(n))
    return randomlist
```

def solinitalaTsp(n) genereaza un vector cu elemente de la 1 la n puse random

Figure 5

```
input: o lista cu valori de la 1 la n puse random
Genereaza 2 valori random intre 0 si n-1 care reprezinta indecsi cu care se vor face swipe
output: se returneaza o copie a vectorul modificat
'''
def generareVecinTsp(solinitTsp):
    randomtwo= random.sample(range(0,len(solinitTsp)), 2)
    copie=solinitTsp.copy()
    copie[randomtwo[0]], copie[randomtwo[1]] = copie[randomtwo[1]], copie[randomtwo[0]]
    return copie

def memorieDistante(listaPct):
    matrice=[[0]*len(listaPct)]*len(listaPct)
    for i in range(len(listaPct)):
        val=[0]*len(listaPct)
        for j in range(i,len(listaPct)):
            val[j]=distEuclidiana(listaPct[i],listaPct[j])
            matrice[i]=val.copy()
    return matrice
```

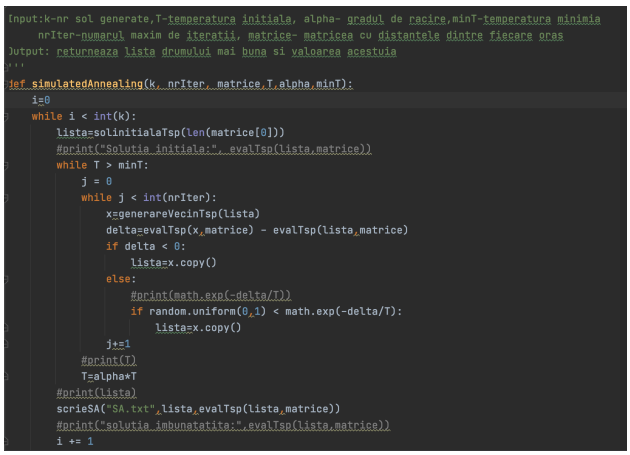
def memorieDistante(listaPct): primeste ca parametrii o lista cu poztile fiecarui punct din spatiu Pune in triunghiul superior dintr-o matrice fiecare distantele dintre toate punctele Pe linia i se pune distanta Euclidiana de la punctul i la cele n-i puncte

Figure 6

```
input: o lista cu valori de la  1-n random puse si o matrice superioara ce are valorile distantei euclidiane dintre pct
Face o suma cu distantele dintre orase in functie de ordinea lor in lista
output: Un numar intreg ce reprezinta valoarea drumului total prin toate orasele + intoarcerea in orasul initial

'''
def evalTsp(lista, matrice):
    suma=0
    for i in range(len(lista)):
        j=i+1
        if j == len(lista):
            if lista[0] > lista[i]:
                suma=suma+matrice[lista[i]-1][lista[0]-1]
            else:
                suma=suma+matrice[lista[0]-1][lista[i]-1]
            return suma
        if lista[i] > lista[j]:
            suma=suma+matrice[lista[j]-1][lista[i]-1]
        else:
            suma = suma + matrice[lista[i]-1][lista[j]-1]
```

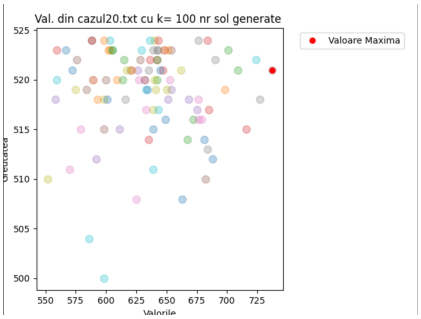
Figure 7

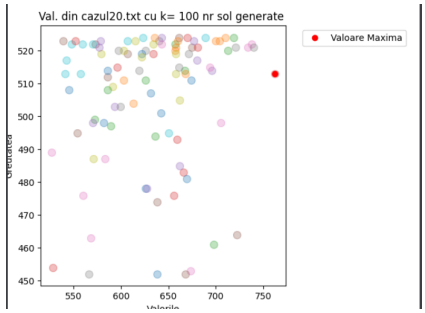


```
Input:k-nr sol generate,T-temperatura initiala, alpha- gradul de racire,minT-temperatura minimia
    nrIter-numarul maxim de iteratii, matrice- matricea cu distantele dintre fiecare oras
Output: returneaza lista drumului mai buna si valoarea acestuia
'''
def simulatedAnnealing(k, nrIter, matrice,T,alpha,minT):
    i=0
    while i < int(k):
        lista=solinitialaTsp(len(matrice[0]))
        #print("Solutia initiala:", evalTsp(lista,matrice))
        while T > minT:
            j = 0
            while j < int(nrIter):
                x=genereareVecinTsp(lista)
                delta=evalTsp(x,matrice) - evalTsp(lista,matrice)
                if delta < 0:
                    lista=x.copy()
                else:
                    #print(math.exp(-delta/T))
                    if random.uniform(0,1) < math.exp(-delta/T):
                        lista=x.copy()
                j+=1
            #print(T)
            T=alpha*T
        #print(lista)
        scrieSA("SA.txt",lista,evalTsp(lista,matrice))
        #print("solutia imbunatatita:",evalTsp(lista,matrice))
        i += 1
```

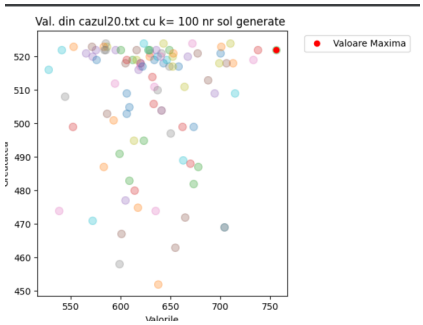
matrice -reprezinta matricea distantelor dintre punctele din spatiu, k -nr de solutii generrate si nrIter- nr de iteratii maxim admise

Cele trei figuri de mai jos reprezinta **Tabu Search** pentru problema rucsacului in instanta cu 20 de obiecte(fisierul: cazul20.txt), unde k rerezinta numarul de solutii generate.





(a) k=100,iteratii=100, tabu=20 (b) k=100,iteratii=100, tabu=20



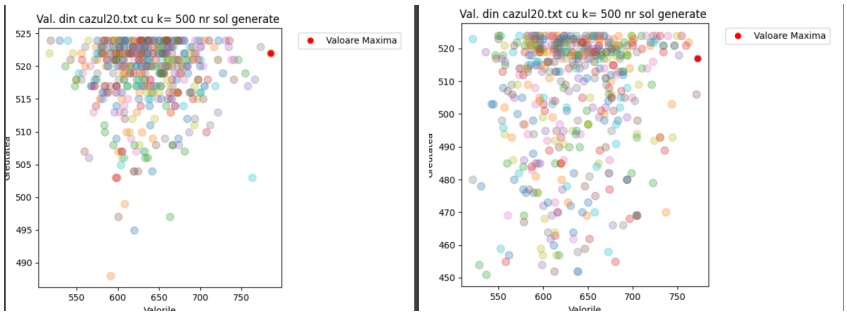
(c) k=100,iteratii=1000, tabu=20

In imaginea (a) sunt reprezentate 100 de solutii cand numarul de iteratii este 100, iar numarul de iteratii tabu este 10.Valoarea maxima este atinsa in punctul cu valoarea 737 si cu greutatea 521. Media valorilor solutilor fiind 633.95.Cu un timp de exectuie: 7.202945709228516 sec
In imaginea (b) sunt reprezentate 100 de solutii cand numarul de iteratii este 100, iar numarul de iteratii tabu este 20.Valoarea maxima este atinsa in punctul cu valoarea 762 si cu greutatea 513. Media valorilor solutilor fiind 631.6. Cu un timp de exectuie: 8.470772981643677 sec
In imaginea (c) sunt reprezentate 100 de solutii cand numarul de iteratii este 1000, iar numarul de iteratii tabu este 20.Valoarea maxima este atinsa in punctul cu valoarea 756 si cu greutatea 522. Media valorilor solutilor fiind

631.38. Cu un timp de exectuie: 26.35385012626648 sec

Comparand cele doua rezultate ((a) si (b))se observa ca atunci cand sunt mai multe iteratii tabu se obtine o valoare maxima mai buna, cu un timp mai mare de executie (dar care nu este foarte mare). Din imaginea (c) se observa cresterea nr de iteratii, dar nu si modificarea celorlalti parametrii nu aduce o imbunatatire a solutiei. Aceasta avand si un timp destul de mare de executie in comparatie cu primele doua Comparand media valorilor cu cele obtinute la **RHC** pentru 100 de solutii generate se observa o imbunatatire la cazul de fata. Timpul fiind ciudat de scrut la RHC, ceea ce inseamna ca am rulat fara muzica si alte lucruri deschise. Dar evident timpul de executie la **TS** este mai mare, acesta verificand istoricul iteratilor tabu.

Cele doua figuri de mai jos reprezinta **Tabu Search** pentru problema rucsacului in instanta cu 20 de obiecte(fisierul: cazul20.txt), unde k rerezinta numarul de solutii generate.



(a) k=500,iteratii=100, tabu=20 (b) k=500,iteratii=100, tabu=20

In imaginea (a) sunt reprezentate 500 de solutii cand numarul de iteratii este 100, iar numarul de iteratii tabu este 10.Valoarea maxima este atinsa in punctul cu valoarea 785 si cu greutatea 522. Media valorilor solutilor fiind 634.876.Cu un timp de exectuie: 18.27291989326477 sec
In imaginea (b) sunt reprezentate 500 de solutii cand numarul de iteratii este 100, iar numarul de iteratii tabu este 20.Valoarea maxima este atinsa in punctul cu valoarea 772 si cu greutatea 517. Media valorilor solutilor fiind 636.462. Cu un timp de exectuie: 15.178250789642334 sec

Crescand numarul de solutii generate creste si valoarea maxima atinsa, comparand cu cazurile precedente se observa o imbunatatire atat in valoarea maxima cat si in media valorilor. Totusi timpul fiind aproape dublu cand s-a cincit numarul de solutii geneerate.

Figure 10: k=1000,iteratii=100,tabu=17

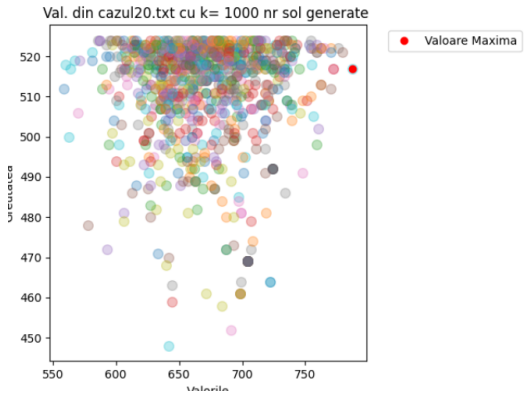
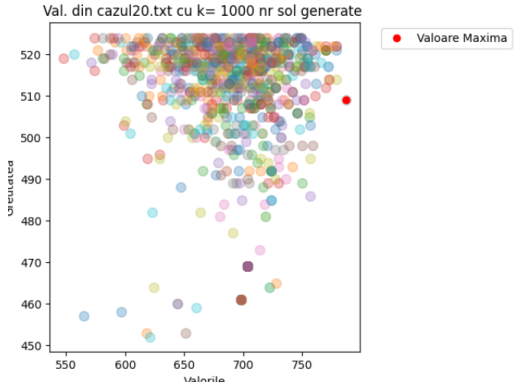


Figure 11: k=1000,iteratii=1000,tabu=17

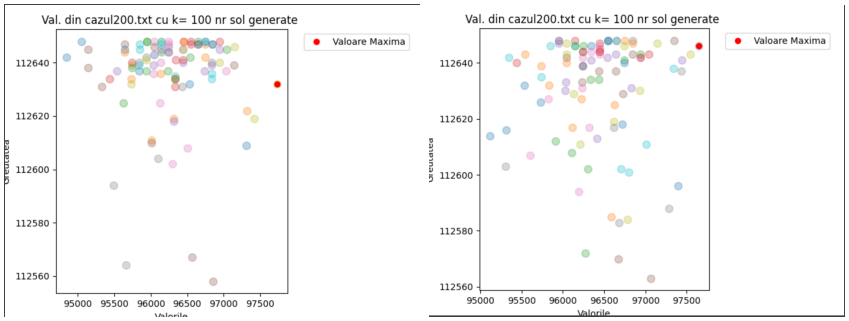


In figura 3 se poate observa reprezentarea grafica pentru **TS** atunci cand numarul de solutii este 1000, numarul de iteratii 100 si nr de iteratii tabu 17.Valoarea maxima este atinsa in punctul cu valoarea 787 si cu greutatea 517. Media valorilor solutilor fiind 671.351.Cu un timp de exectuie:19.913827180862427 sec.
In figura 4 se poate observa reprezentarea grafica pentru **TS** atunci cand

numarul de solutii este 1000, numarul de iteratii 1000 si nr de iteratii tabu 17.Valoarea maxima este atinsa in punctul cu valoarea 787 si cu greutatea 509. Media valorilor solutilor fiind 692.348.Cu un timp de exectuie:63.96424698829651 sec.

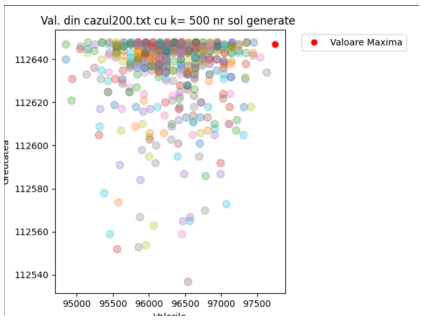
Comparand datele de la figura 3 si 4 se observa cum s-a imbunatatit doar valoarea medie atunci cand s-a crescut numarul de iteratii la 1000, iar timpul de executie fiind destul de mare si nu obtine o valoare mai buna.

Cele doua figuri de mai jos reprezinta **Tabu Search** pentru problema rucsacului in instanta cu 200 de obiecte(fisierul: cazul200.txt), unde k rerezinta numarul de solutii generate.

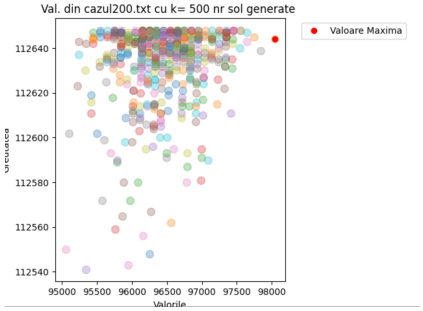


(a) k=100,iteratii=20, tabu=10 (b) k=100,iteratii=40, tabu=20

In imaginea (a) sunt reprezentate 100 de solutii cand numarul de iteratii este 20, iar numarul de iteratii tabu este 10.Valoarea maxima este atinsa in punctul cu valoarea 97732 si cu greutatea 112632. Media valorilor solutilor fiind 96269.51.Cu un timp de exectuie: 96.79393005371094 sec.
In imaginea (b) sunt reprezentate 100 de solutii cand numarul de iteratii este 40, iar numarul de iteratii tabu este 20.Valoarea maxima este atinsa in punctul cu valoarea 97646 si cu greutatea 112646. Media valorilor solutilor fiind 96415.42. Cu un timp de exectuie: 90.33718395233154 sec.
Comparand cele doua cazuri se observa ca valoarea maxima nu se imbunatateste neaparat, dar media valorilor de imbunatateste atunci cand nr de iteratii tabu creste. Timpul de rulare nu difera foarte mult intre ele.

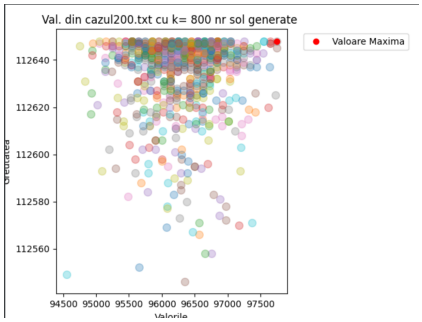


(a) k=500,iteratii=50, tabu=10

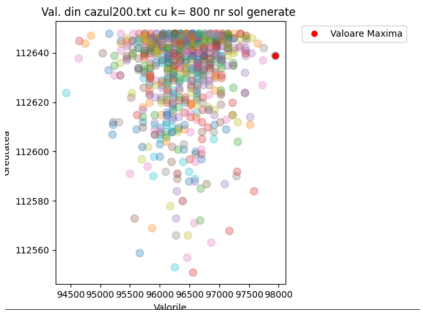


(b) k=500,iteratii=50, tabu=20

In imaginea (a) sunt reprezentate 500 de solutii cand numarul de iteratii este 50, iar numarul de iteratii tabu este 10.Valoarea maxima este atinsa in punctul cu valoarea 97747 si cu greutatea 112647. Media valorilor solutilor fiind 96326.156.Cu un timp de exectuie: 554.1479880809784 sec.
In imaginea (b) sunt reprezentate 500 de solutii cand numarul de iteratii este 50, iar numarul de iteratii tabu este 20.Valoarea maxima este atinsa in punctul cu valoarea 98044 si cu greutatea 112644. Media valorilor solutilor fiind 96437.25. Cu un timp de exectuie: 543.0967481136322 sec.
Comparand cele doua cazuri se observa ca valoarea maxima si media valorilor se imbunatateste atunci cand interatile tabu cresc impreuna cu numarul de solutii generate.



(a) k=800,iteratii=40, tabu=10



(b) k=800,iteratii=40, tabu=20

In imaginea (a) sunt reprezentate 500 de solutii cand numarul de iteratii este 50, iar numarul de iteratii tabu este 10.Valoarea maxima este atinsa in punctul cu valoarea 97748 si cu greutatea 112648. Media valorilor solutilor fiind 96300.4525.Cu un timp de exectuie: 653.9635739326477 sec.
In imaginea (b) sunt reprezentate 500 de solutii cand numarul de iteratii

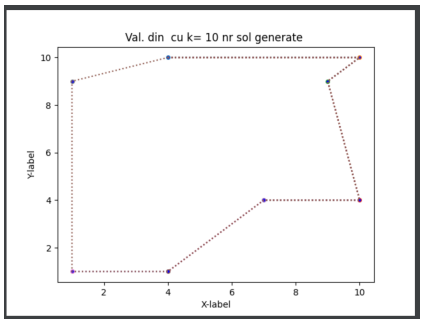
este 50, iar numarul de iteratii tabu este 20.Valoarea maxima este atinsa in punctul cu valoarea 97939 si cu greutatea 112639. Media valorilor solutilor fiind 96408.26. Cu un timp de exectuie: 596.2787129878998 sec.

Comparand toate cazurilea de **Tabu Search** se observa ca de la un anumit numar de solutii generate(pentru o anumita instanta) nu se mai observa imbunatatiri asa bune cand vine vorba de cea mai buna valoare, iar timpul de executie creste considerabil.

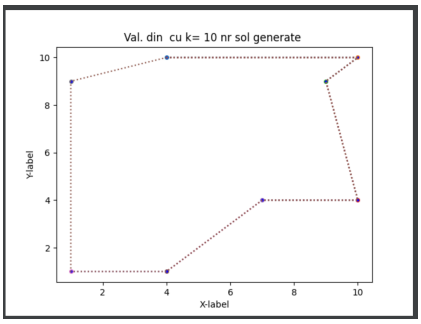
TABU SEARCH VS RANDOM HILL CLIMBING Observ o imbunatatire din punct de vedere al solutilor obtinute la Tabu Seach deoarece pentru un numar mai mic de solutii generate se obtin valori mai bune decat la RHC cand sunt generate mult mai multe solutii. De exemplu la TS pentru 100 de solutii generate cu un nr de 20 de iteratii si 10 interatii tabu s-a obtinut o valoare maxima mai mare decat la RHC cand s-au generat 10000 de solutii cu 1000 de incercari. De asemenea media valorilor este mai buna la TS decat la RHC. Toate acestea platind un timp de executie mai indelungat.

SIMULATING ANNEALING TSP

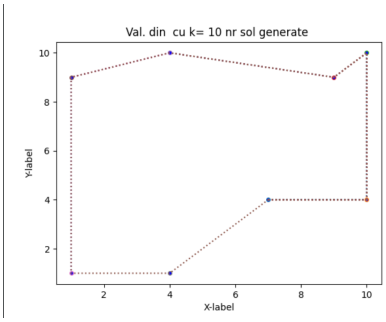
Imaginile de mai jos reprezinta **simulated annealing tsp** pentru o lista ce contine urmatoarele puncte: [(1,1),(4,1),(7,4),(10,4),(10,10),(9,9),(4,10),(1,9)]. Fiecare reprezinta cea mai buna solutie dintre 10 rulari.



(a) [7, 5, 6, 4, 3, 2, 1, 8]



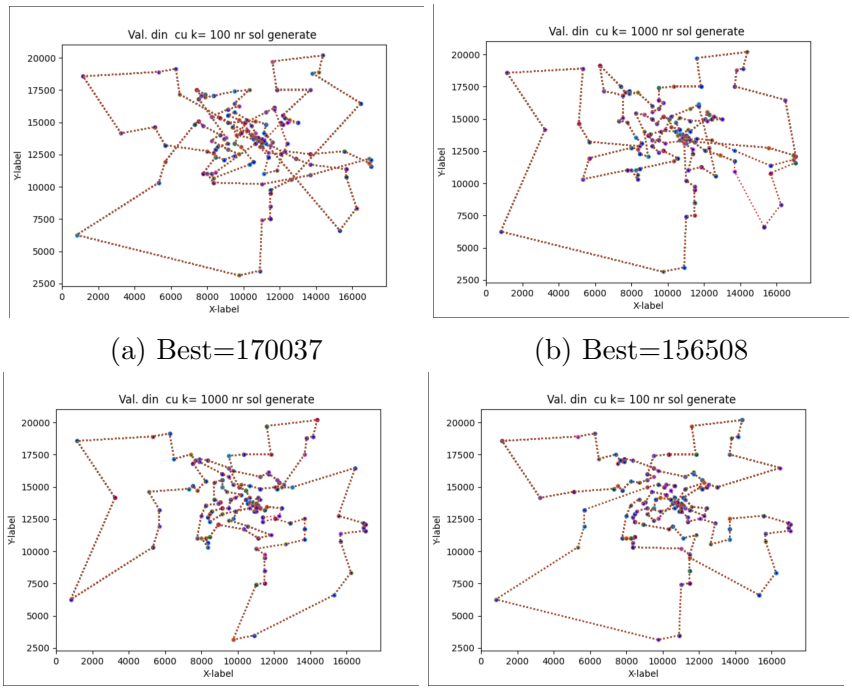
(b) [1,2,3,4,5,6,7,8]



(c) [8,1,2,3,4,5,6,7]

In imaginea (a) T = 10000,alpha = 0.9999, minT = 0.0001, numarul de solutii generate:10, numarul de iteratii a fost 3, cel mai buna distanta a fost de 33 cu traseul [7, 5, 6, 4, 3, 2, 1, 8], intr-un timp de 605.6898491382599 sec. Media:33.0
In imaginea (b) T = 1000, alpha = 0.777, minT = 0.0001, numarul de solutii generate:10, numarul de iteratii a fost 3, cel mai buna distanta a fost de 33 cu traseul [1,2,3,4,5,6,7,8], intr-un timp de 0.05170083045959473 sec.Media:33.0
In imaginea (c)T = 100, alpha = 0.555, minT = 0.001 numarul de solutii generate:10, numarul de iteratii a fost 4, cel mai buna distanta a fost de 33 cu traseul [8,1,2,3,4,5,6,7], intr-un timp de 59.26865100860596 sec.Media:35.0
Se observa ca atunci cand timpul de racire este mare se obtine o solutie foarte buna.

In imaginile de mai jos este reprezentarea grafica a problemei TSP rezolvata cu algoritmul Simulated Annealing pentru fisierul bier127.txt. Acestea reprezinta bestul din 10 rulari cu parametrii aferenti din tabelul de mai jos.



(a) Best=170037 (b) Best=156508

(c) Best=134623 (d) Best=135305

Figure 17: BIER127

SIMULATED ANNEALING TSP BIER127

T	ALpha	MinT	nrSol Generate	nr Maxim iter	nr RULARI	TIMP	Media	Best
1000	0.999	0.0001	100	3	10	52.08837199	184377.5	170037
1000	0.999	0.0001	1000	10	10	141.0551949	169880.1	156508
1000	0.9999	0.0001	1000	10	10	1765.975122	143351	134623
10000	0.999	0.0001	100	100	10	1575.048547	141883.4	135305

Se observa ca atunci cand timpul de racire este destul de mic nu se obtin niste rezultate asa bune.Acest lucrtu depinde si de numarul de iteratii maxime, daca e mic acesta nu accepta asa multi vecini mai rai, asteptand sa scada exp(-delta/T). Faptul ca nu ne permite sa acceptam foarte multe

solutii mai slabe restrictioneaza spatiul de cautare si ajunge blocat in minime locale.
Comparand ultimele doua valori din table, acolo unde nr de iteratii maxime este mai mare, valoarea medie este mai buna. Valoare cea mai buna dintre ele nu difera foarte mult. In asa mod se reuseste o cunoastere a spatiului mai buna.