Flow-shop ütemezés

Genetikus algoritmus
segítségével

2022/23 I félév

Készítette: ***Oravecz Áron***

Neptun kód: ***D3U3EE***

Feladat leírás:

A flow-shop feladat egy ún. egyutas, többoperációs gyártásütemezési feladat. A gyártásütemezési feladatok formális leírásra az alfa, béta, gamma jelölést alkalmazzuk ($a|b|g$). E szerint a rendszer szerint az egyszerű flow-shop feladat leírása:

$$F||C_{max}$$

A feladatomat inicializálnom kellett valamilyen véletlen számgenerátorral, illetve a munkák és a gépek száma különböző méretűeknek kellett lennie.

A legenerált feladatokat perzisztens vagy seed-hez kötéses procedúrával kellett elkészítenem.

Az genetikus algoritmus segítségével kitudtam számolni a legoptimálisabb ütemezési sorrendet és a hozzá tartozó C_max értéket.

A Flow-shop problémája és a genetikus algoritmus elmagyarázása a „feladatleiras.pdf" -ben megtalálható.

Célszerűnek tartottam a feladat megoldásához használni a Python 3.11 -es verzióját. Illetve könyvtárnak a Numpy-t.

A Numpy egy kiegészítő a Python-hoz, ami a többdimenziós tömbök és mátrixok használatát támogatja egy nagy magas szintű matematikai függvénykönyvtárral.

Hátránya:

- Lassabb lefutási idő, mert nagyobb memória igénylés
    - Ideiglenes tömbök létrehozása → több bemenet

Feladat megoldásom menete:

1. Importálom a lehetségesen használt könyvtárakat. A többit utólagosan hozzáadtam, hogy hibamentesen fusson a program.

```
from random import choices, randint, randrange, random
import time, copy
import numpy as np
import random
```

2. Létrehoztam egy „main" függvényt, hogy majd az általam definiált függvényeket itt meghívhassam és a lokális változókkal a paraméterlistát kitölthessem.

Változók:

```
def main():

    n = 10 #munkák száma
    m = 5 #gépek száma
    seed = 0 #véletlen számgenerátor állapotának beállításához szükséges változó
    seed2=42

    popmeret= 10 #populáció mérete
    genszama = 5 #generáció száma/mennyisége
    mutszama = 2 #mutáció száma/mennyisége

    #Gantt diagram
    ProcT = 0 # ProcT = műveleti idő
    StarT = 0 # StartT = indítási idő
    EndT = 0  # EndT = befejezési idő
```

Függvény meghívások:

```
    mat = np.array(FlowShopMatrix(n,m,seed))
    print("Szimuláció: ","\n",mat)

    s = utemtervRand(n,seed2)
    print("Ez az ütemterv:",s)

    print(CmaxSzamolasa(m,n,mat,s))

    print(mutacio(s))

    print(genetic(s,popmeret,genszama,mutszama,m,n,mat))

  #  print(print_Job_Gantt(mat,n,m,s))

if __name__=="__main__":
    main()
```

3. Első legfontosabb lépés a kezdetleges ütemezés és a munkánk véletlenszerű legenerálása.

```
def FlowShopMatrix(n,m,seed):
    np.random.seed(seed)
    return np.random.randint(1,500, size=(n,m))
#öröklődés miatt kell, seed alapján egy fajta randomizált értéket add vissza mátixba
```

Ez a függvény az ütemterv szimulációjának az időadatait fogja feltölteni a dinamikus mátrix mérete alapján. A mátrixunk mérete a gépek száma és a munkák száma alapján változtatható.

Ennek a függvénynek a visszatérési értéke egy mátrix, ami a Numpy randomszám generátor segítségével töltöttem fel.

Példa a futtatási időből:

```
Szimuláció:
 [[173  48 118 193 324]
 [252 196 360  10 212]
 [278 243 293  88  71]
 [473  89 397 315 194]
 [487  40  88 175  89]
 [338 166  26 334  73]
 [266 405 116 465 244]
 [198 336 432 449 339]
 [100 473 178 244 286]
 [148 148 399 424 289]]
```

Az ütemterv generálása is hasonló módon történt, mint az előző mátrix. Csak itt véletlenszerűen kellett **választani(choice)** a gépek közül, hogy létrehozzam a véletlen sorrendű ütemtervet, amit ezek után ki kellett értékelni.

```
def utemtervRand(n,seed2):
    np.random.seed(seed2)
    s = np.random.choice(range(n),n,replace=False)
    return s
#ugyan az mint a FlowShopMatrix
```

Példa a futtatási időből

```
Ez az ütemterv: [8 1 5 0 7 2 9 4 3 6]
```

4. Miután meglett az ütemtervünk és a mátrixunk elkezdhetjük ennek a C-max eredményét kiszámolni. Ehhez külön egy függvényt írtam, ami: „**Cmaxszamolasa**” névvel definiáltam melyhez paraméterként társítottam a gépek számát(**n**), munkák számát(**m**), a véletlen számgenerátorral létrehozott mátrixot, amit elmentettem egy változóba (**mat**), és az ütemtervet(**s**). Először eltárolom a kezdő és befejezési munkák idő értékét. természetesen mikor létrehozom ezeket a listákat két for ciklus segítségével és kinullázom őket. Majd egy cost(érték) ami végig megy a gépeken. Majd dupla forciklussal végigmegyek először a munkákon utána a gépeken a C_max értékét egyenlővé teszem a gépek értékének j-dik elemével. Majd elágazás segítségével eldöntöm, ha az adott indexű elem nagyobb, mint nulla akkor a C_max értéke a legnagyobb értékre változtatom a cost listából. majd beállítom a c_max értékét úgy, hogy összeadom a c_max-ot a véletlen generált mátrixunk ütemterve szerinti utolsó elemével, ami az i-dik indexen van. ezek után könnyen ki tudjuk számolni a befejezési és

kezdeti munkának az értékét. C_max értéke pedig az utolsó befejezési munka értéke lesz, mert a ciklus végére mindig az marad. Próbálkoztam visszatérési értéknek megadni több elemet is a befejezést és a kezdést, de a generációs algoritmusba problémás lett volna megoldani, így az egyszerűbb utat választottam.

```
def CmaxSzamolasa(m,n,mat,s):
    kezdmunka = [[0 for x in range(n)] for y in range(m)]
    befmunka = [[0 for x in range(n)] for y in range(m)]
    cost = [0 for i in range(n)] #érték végigmegy a gépeken

    #print("kezdeti munka: \n",kezdmunka)
    #print("befejezési munka: \n: ",befmunka)
    #print("érték: \n: ",cost)

    for i in range(m): #munkákon megy végig majd
        for j in range(n): #gépeken
            c_max = cost[j]
            #print("C_max értéke most = ",c_max)

            if j > 0:
                c_max = max(cost[j-1], cost[j])

            cost[j] = c_max + mat[s[j] - 1][i]
            #print(j,". elem értéke = ",cost[j])


            befmunka[i][j] = cost[j]
            #print("befejezési munka: ",befmunka[i][j]) #ez mindig annyi mint a cost j. eleme

            kezdmunka[i][j] = befmunka[i][j] - mat[s[j] - 1][i]

            #print("A kezdeti munka kivonással = ",kezdmunka[i][j])

            c_max = befmunka[i][j]

    print("\n \n \n")
    print("C max értéke: ",c_max)
    print("Kezdeti érték = \n",kezdmunka)
    print("Befejezési érték = \n",befmunka)
    print("\n \n \n")

    return c_max
#    return{
#        "c_max": c_max,
#        "Kezdeti értékek ": kezdmunka,
#        "Befejezési értékek ": befmunka
#        }
```

futtatási eredmény:

```
C max értéke:  3752
Kezdeti érték =
 [[0, 198, 371, 858, 1006, 1272, 1524, 1624, 2097, 2375], [198, 534, 858, 1006, 1272,
1677, 1873, 2346, 2435, 2713], [534, 966, 1084, 1172, 1677, 1873, 2346, 2524, 2921, 32
14], [966, 1415, 1608, 1783, 2207, 2672, 2682, 2926, 3241, 3329], [1415, 1754, 2078, 2
207, 2672, 2916, 3128, 3414, 3608, 3679]]
Befejezési érték =
 [[198, 371, 858, 1006, 1272, 1524, 1624, 2097, 2375, 2713], [534, 582, 898, 1154, 167
7, 1873, 2346, 2435, 2678, 2879], [966, 1084, 1172, 1571, 1793, 2233, 2524, 2921, 3214
, 3240], [1415, 1608, 1783, 2207, 2672, 2682, 2926, 3241, 3329, 3663], [1754, 2078, 21
67, 2496, 2916, 3128, 3414, 3608, 3679, 3752]]
```

5. Ezután elkezdhetem a speciális genetikus feladatomat.

$$F||C_{max}$$

Jelen esetben egy olyan gyártásütemezési feladatott kell megoldanom, ahol a gépek között a munkák várakozhatnak, vagyis az operációk kötött és minden munka esetében azonos. A legfontosabb megtudni melyik a legjobb megoldás a befejezési időpont kiszámolására, vagyis úgy kalkulálni, alakítani, hogy munka befejezési időpontja minimalizálva legyen. Ehhez szükségem lesz egy mutáció és egy genetikus függvényre, amivel megváltoztatom a ütemtervemet és olyan generációkat készítek ami egy bizonyos lefuttatási idő alatt megtalálja a legoptimálisabb értéket.

6. Mutáció függvényem lényege, hogy az idáig használt ütemterv egy olyan fajta verzióját készítsem el, ami eltér az előzőtől és ezt véletlen számgenerátorral tegyem meg. A mutánsom úgy nézz ki, hogy két index elemét felcserélem véletlenszerűén. Ha lenne rá esély, hogy ugyan olyan számot generáljon számunkra a random metódus ezt egy while ciklussal megoldható.

```
def mutacio(s):
    temp=0
    mutans = copy.deepcopy(s)
    b = random.randint(0,len(mutans)-1)
    a = random.randint(0,len(mutans)-1)
    while a == b:
        b = random.randint(0,len(mutans)-1)

    mutans[a],mutans[b]=mutans[b],mutans[a]

    return mutans
```

Gyakorlati tanárom: Fazekas Levente. Az ő weboldalán található genetikus algoritmusban hasonló mutáció függvény ismerhető fel, de ezt a struktúrát felhasználva tovább fejlesztettem és alkalmaztam az én programomban.

7. Genetikus függvényem lényege, hogy egy bizonyos populáció méret, generáció- és mutáció száma alapján kiértékelhessem a **legjobb megoldást,** és **a legjobb c_max értéket** (ami a legjobb egyed ideje lesz).

```
def genetic(s,popmeret,genszama,mutszama,m,n,mat):
    populacio =[list(s)]
    for i in range(popmeret - 1):
        populacio.append(mutacio(s))

    best_megold = max(populacio, key=lambda ch: CmaxSzamolasa(m,n,mat,ch))
    best_c_max = CmaxSzamolasa(m,n,mat,best_megold)

    for j in range(genszama):
        for i in range(mutszama):
            populacio.append(mutacio(populacio[random.randint(0,len(populacio)-1)]))

        populacio = sorted(populacio, key = lambda ch: CmaxSzamolasa(m,n,mat,ch))
        populacio = populacio[:popmeret] #levágjuk a méretét a kívántra

        best_egyed = populacio[0]
        best_egyed_ido = CmaxSzamolasa(m,n,mat,best_egyed)

        if(best_egyed_ido < best_c_max):
            best_c_max = best_egyed_ido
            best_megold = best_egyed

    return best_megold
```

A függvény definiálása során a populációmat külön ütemterv szerinti listába hozom létre. Majd végig megyek a populáció mérete szerint a populáción és hozzá adom a mutációt. Ezután a legjobb megoldást kiszámolom a C-maxra, amihez alkalmazom a „CmaxSzamolas” függvényemet is, és ezt az értéket eltárolom a legjobb megoldásba. majd a legjobb c_maxot kiszámolom a legjobb megoldások szerint. Végül a generációk és mutációk száma alapján végig megyek a populációmon rendezem őket és levágom a méretét a kívántra és visszaadom elágazás alapján a lejobb megoldás értékét.

Futtatás során az utolsó C_max értéke, Kezdeti- és Befejezési értékek

```
C max értéke:  3732
Kezdeti érték =
 [[0, 173, 371, 858, 1006, 1272, 1524, 1624, 2097, 2375], [173, 371, 858, 1006, 1272, 1677
, 1873, 2346, 2435, 2713], [221, 707, 1139, 1227, 1677, 1873, 2346, 2524, 2921, 3214], [33
9, 1139, 1588, 1763, 2187, 2652, 2662, 2921, 3236, 3324], [532, 1588, 1927, 2187, 2652, 28
96, 3108, 3394, 3588, 3659]]
Befejezési érték =
 [[173, 371, 858, 1006, 1272, 1524, 1624, 2097, 2375, 2713], [221, 707, 898, 1154, 1677, 1
873, 2346, 2435, 2678, 2879], [339, 1139, 1227, 1626, 1793, 2233, 2524, 2921, 3214, 3240],
 [532, 1588, 1763, 2187, 2652, 2662, 2906, 3236, 3324, 3658], [856, 1927, 2016, 2476, 2896
, 3108, 3394, 3588, 3659, 3732]]



[1 8 5 0 7 2 9 4 3 6]
```

Felhasznált irodalmak:

https://www.w3schools.com/python/

https://ai.leventefazekas.hu/lessons/2022-10-18-genetic-algorithms/

https://hu.wikipedia.org/wiki/NumPy

https://pythonidomar.wordpress.com/python-parancsok-magyar-jelentes/