

Jegyzőkönyv

Mesterséges Intelligencia

Féléves feladat

Flow-shop ütemezés, Genetikus Algoritmus

Készítette: Juhász Ákos

Neptunkód: F58KQ8

Tartalom:

1. Feladat ismertetése:	2
2. Választott specifikációk	2
3. Program működése	2
4. Program felépítése	4
□ FlowShop.java:	4
□ GeneticAlgorithmm.java:	4
□ StoreMatrix.java:	5
5. Futási eredmények	6

1. Feladat ismertetése:

A fél éves beadandó témája a flow-shop ütemezés végrehajtása, genetikus algoritmus használatával.

A program futásához szükséges legelső adatokat a program perzisztensen tárolja. A program futhat a megadott maximum 500*200-as mátrix segítségével, vagy random adatok generálására is meghívható. Az adatok beolvasása során az egy létrejövő generációhoz szükséges permutációkat több mutációs és egy rekombinációs módszerrel hozzuk létre. A generációból létrejött fenotípusban a „kihalás” folyamatát konstans segítségével végezhető, de a generáció szimpla felezésére is van lehetőség.

A program minden futásnál az adott 500*20-as mátrixból képes kisebbeket képezni, és indítás után többször is végighalad a teljes folyamat. A program futásának végét a „Run over!” felirat jelzi.

2. Választott specifikációk

A beadandó java nyelven készült, Eclipse Ide környezetben. A program alapvetően konzolra történő kiíratásra képes.

3. Program működése

A program main metódusát a GeneticAlgorithmm.java osztály tartalmazza, meghívásakor létrehoz egy arraylistet integer típusú kétdimenziós tömböknek, ebben fogjuk tárolni a generációk adatait. Minden egyes flow-shop futtatását

egy kétdimenziós arrayban tároljuk melyek J és M+1 hosszúak. A jobok hosszát tartalmazó array elejét mindig megtoldjuk egy egyesével növekvő integerrel, ez tesztelésnél és eredmény kiíratáskor segít nekünk szemmel tartani vagy szemléltetni a jobok sorrendjét.

Az alapadatok egy 500*20-as mátrixban vannak tárolva, ezeket egy metódussal kisebb mátrixokra bonthatjuk. A main metódus folyamán ezekre a kisebb, illetve eredeti mátrixokon végezzük egy megfelelő műveleteket. A generikus algoritmus meghívásáért a runOnegeneration() metódus felelős. Egy for cikluson belül hajtódnak végre a metódusok, ami addig fut, amíg az el nem éri a megadott generációs számot.

1. Először feltöltjük a generációt mutáción keresztül. Random szám generálással megadva 50% eséllyel hívhatunk meg két mutációs metódust. Egyik lehetőség szerint két random J sorrendjét cseréli fel, a másik szerint pedig egy véletlen méretű szakaszt kiválasztunk a mátrixban, amik a J-k sorrendjét tartalmazzák. Majd azok sorrendjét megfordítjuk.
2. Meghívjuk a generáció minden tagjára a rekombinációs metódust. Ekkor véletlenszerűen kiválasztunk egy másik mátrixot a generációban, majd kiválasztunk egy véletlen hosszúságú szakaszt és megcseréljük a kettő között őket. A csere úgy történik, hogyha mondjuk a) szakasz első eleme egy 2-es b) szakaszé pedig egy 3-as, akkor b) szakaszban megkeressük és megcseréljük a 2-es és 3-as számjegyet, majd hasonlóan teszünk az a) szakasszal is.
3. Ezután kiszámoljuk és egy arrayban letároljuk minden flow-shop futtatás befejező értékét, az az a C_{\max} értéket. Ezt felhasználva növekvő sorrendben rendezzük a generáció elemeit C_{\max} alapján, így létrehozva a fenotípust.
4. A „kihalás” folyamatát konstans alapú módszerrel hajtjuk végre. A feladatkiírásban adott képlet annyiban módosítva lett, hogy az első tag mindig túléli, hogy a legjobb eredményünket ne veszíthessük el.

$$P_1 = P_c$$

$$P_2 = (1 - P_c) \cdot P_c$$

$$P_3 = (1 - P_c)^2 \cdot P_c$$

$$P_{n-1} = (1 - P_c)^{n-2} \cdot P_c$$

$$P_n = (1 - P_c)^{n-1},$$

Gyakorlatban ez annyit tesz, hogy P1 túlélési értéke mindig 1, és az igazi számítás P2-től kezdődik, aminek a túlélési értéke Pc. A választott konstans értéke 0.8.

(Olyan metódus is implementálva van, ami egyszerűen kitörli a fenotípus rosszabb felét. A két módszer hasonló eredményekre jut, de én a konstans alapúnál maradtam.)

5. Miután a for ciklus eléri a megadott generációk számát a generikus algoritmus leáll. Végül a metóduson kívül kiszámoljuk és kiíratjuk másodpercekben az adott mátrixhoz tartozó futási időt.
6. Pár funkció nincsen használva normál futtatási körülmények között, ide tartozik:
 - Mátrix kirajzolása
 - Gantt diagram szerű kiíratása a mátrixnak
 - Random értékekkel feltöltött mátrix generálása

4. Program felépítése

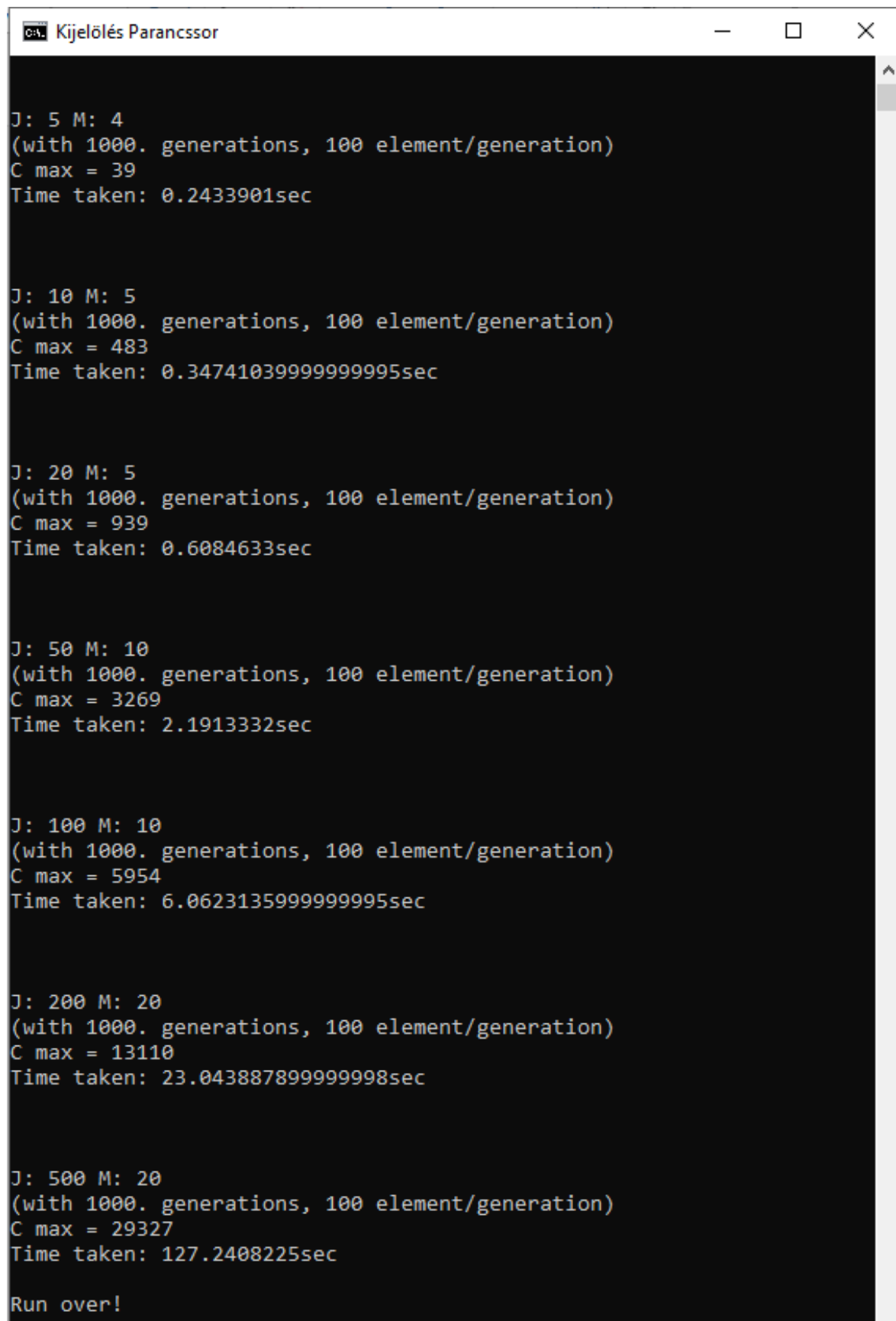
A program 3 osztályra van bontva:

- FlowShop.java:
 - Magáért a flow-shop folyamatért felel, illetve egyéb (mint például mátrix kiíratás) metódusokért.
 - generateRandomData(): adott M és J érték után létrehoz egy feldolgozásra kész arraylistet.
 - calculateResultArray(): visszaadja, hogy minden egyes Machine mikor végzett a rá kiadott Jobokkal.
 - drawMatrixAsNormalData(): a feladatban használt nagy mennyiségű adat legenerálásához szükséges, nem vesz részt a folyamatban.
 - drawMatrix(): kirajzolja egy flow-shop folyamat adatait mátrix formában.
 - drawDiagram(): a mátrixot csak parancssorra kiírja egy Gantt diagramot idéző formátumban.
- GeneticAlgorithm.java:
 - A genetikus algoritmushoz szükséges metódusokat tárolja, illetve a program main metódusa is innen fut.
 - runOnegeneration(): egy generáció végig menő folyamatokhoz szükséges metódusok meghívásáért felel.
 - generateGeneration(): létrehoz egy generációt, és hozzáadja az alap mátrixot illetve annak a fordítottját.

- fillGeneration(): 50%-os eséllyel választ a két mutációs metódus között, azokat hívja meg egy mátrix minden sorára, majd hozzáadja a generációhoz.
- mutateByReplace(): a két mutációs metódus egyike. Két random elemet kiválaszt, és megcseréli a sorrendjüket.
- mutateByBackwards(): egy véletlen, a J-k számának felénél nem nagyobb szakaszt kiválaszt a mátrixban és megfordítja a sorrendet benne.
- recombination(): két mátrix között kicserél egy véletlenszerűen generált hosszúságú szakaszt.
- callRecombination(): meghívja a rekombinációt a generáció elemeire.
- getCMax(): a generáció összes elemének kiszámolja a C_{\max} értékét és eltárolja azt egy arrayban.
- generatePhenotype(): C_{\max} alapján növekvő sorrendbe rendezi a generáció elemeit.
- extinctionSecondHalf(): eldobja a generáció lassabbik felét. Jelenleg a konstans alapú módszer van használatban helyette.
- extinctionConstant() : konstans alapú „kihalás”-ért felelős metódus, de a legjobb elem mindig túléli. A konstans értéke alaptól 0.8.
- StoreMatrix.java:
 - Csak az alapadatok tárolásért, illetve azok megfelelő formára hozzászáért felel.
 - generateSmallerData(): az alap 500*20-as mátrixot megadott M és J érték alapján kisebb mátrixra redukálja.
 - getStoredData(): a tárolt 500*20 integer adatot egy arrayba helyezi, megfelelő formátum alapján.

5. Futási eredmények

Egy futtatás eredményei windows parancssorban futtatva:



```
Kijelölés Parancssor

J: 5 M: 4
(with 1000. generations, 100 element/generation)
C max = 39
Time taken: 0.2433901sec

J: 10 M: 5
(with 1000. generations, 100 element/generation)
C max = 483
Time taken: 0.34741039999999995sec

J: 20 M: 5
(with 1000. generations, 100 element/generation)
C max = 939
Time taken: 0.6084633sec

J: 50 M: 10
(with 1000. generations, 100 element/generation)
C max = 3269
Time taken: 2.1913332sec

J: 100 M: 10
(with 1000. generations, 100 element/generation)
C max = 5954
Time taken: 6.0623135999999995sec

J: 200 M: 20
(with 1000. generations, 100 element/generation)
C max = 13110
Time taken: 23.043887899999998sec

J: 500 M: 20
(with 1000. generations, 100 element/generation)
C max = 29327
Time taken: 127.2408225sec

Run over!
```