Dokumentáció

Mesterséges intelligencia alapok

Féléves feladat

Flow-shop ütemezési probléma Genetikus algoritmussal

Készítette: Demján Csongor

Neptunkód: SZ7MGG

Dátum: 2022/11/24

Választott nyelv: Python

A feladat forráskódja a SZ7MGG\_mesint.py néven található meg. Az adatok beolvasására a base\_data.csv file-t, a log.txt file-t az eredményeket kiolvasására használtam. A dokumentáció a kódot logikailag feldarabolva magyarázza.

Szükséges Importok

import csv  
import random

Az excel-ből olvasásáshoz szükséges: csv

Random számok generálásához: random

Adatok beolvasása

Beolvassuk a base\_data.csv file adatait, az első sor kihagyásával. A függvénnyel egy listában lévő listát viszünk tovább.

def filereader(file\_name):  
 data\_list = []  
 with open(file\_name, 'r') as f:  
 reader = csv.reader(f)  
 for count, row in enumerate(reader):  
 if count != 0:  
 data\_list.append(row)  
  
 return data\_list

Munkák feltöltése random számokkal

Megnyitjuk a log.txt file-t, amit feltöltünk a gépek és munkák száma által meghatározott véletlenszerűen generált mátrixxal.

def randomizejobs(machines, jobs):  
 file\_output = open("log.txt", "a")  
 job\_list = [[0 for \_ in range(machines)] for \_ in range(jobs)]  
  
 for i in range(machines):  
 for k in range(jobs):  
 job\_list[k][i] = random.randint(1, 100)  
 file\_output.write(str(job\_list[k][i]) + "\t")  
 file\_output.write("\n")  
 file\_output.close()  
 return job\_list

A genetikus algoritmus

A függvénynek átadjuk a korábban generált adatokat. Létrehozunk egy listát, amely az új generációkat fogja tárolni. Kimásoljuk az előre meghatározott adatokat az előbbi listánkba, miközben számban tartjuk ezen listák sorrendjét.

Elindítjuk a mutációt minden egyes listán és eltároljuk. Az itt meghívott függvény a „Mutáció” címszó alatt található.

A következő lépésben a „Rekombináció” -t hajtjuk végre, szintén minden listán. A ciklusunk közben az 1-es szülő az éppen aktuális lista, a 2-es szülő pedig a következő listát reprezentálja. Az utolsó rekombinációban az 1-es szülő az utolsó lista, a 2-es szülő pedig az első.

Ezt követően a „Fitness” értékeket számoljuk.

Rendezzük idő szerint a listát, majd meghatározunk egy konstanst a túlélési valószínűségre, ez a 0.42 értéket kapta a példában. Létrehozunk egy véletlenszerű számot 0.0 és 1.0 között.

Az utolsó lépésben azt vizsgáljuk és íratjuk a log.txt-be, hogy mely generációk élték túl, és mennyire „jók”. A legjobbtól a legrosszabbig, 3 részre bontjuk az eredményeket, amiket kaphatunk.

def get\_new\_genetic(machines, jobs, job\_list, base\_list, generations):  
 actual\_data = [[0 for \_ in range(jobs)] for \_ in range(generations)]  
 time = []  
 order\_of\_lists = []  
  
 for i in range(generations):  
 for k in range(jobs):  
 actual\_data[i][k] = base\_list[k]  
 time += [0]  
 order\_of\_lists += [i]  
  
 for i in range(generations):  
 actual\_data[i] = mutation(actual\_data[i], jobs)  
  
 for i in range(generations - 1):  
 actual\_data[i] = recombination(actual\_data[i], actual\_data[i + 1], jobs)  
  
 actual\_data[generations - 1] = recombination(actual\_data[generations - 1], actual\_data[0], jobs)  
  
 for i in range(generations):  
 time[i] = fitness(machines, jobs, job\_list, actual\_data[i])  
  
 time, order\_of\_lists = sort\_list(time, order\_of\_lists, generations)  
  
 probability = 0.42  
  
 random\_number = random.random()  
  
 for i in range(generations):  
 if i == 0:  
 if random\_number < probability:  
 return actual\_data[order\_of\_lists[i]], time[i]  
 else:  
 random\_number -= probability  
 else:  
 if random\_number < (pow(1 - probability, i) \* probability):  
 return actual\_data[order\_of\_lists[i]], time[i]  
 else:  
 random\_number -= (pow(1 - probability, i) \* probability)  
  
 return actual\_data[order\_of\_lists[generations - 1]], time[generations - 1]

Mutáció

Itt gyakorlatilag csak annyi történik, hogy az ’x’, és ’y’ változók random kapnak egy számot, majd ezeket megcseréljük a listánkban.

def mutation(actual\_data, jobs):  
 x = random.randint(0, jobs - 1)  
 y = random.randint(0, jobs - 1)  
  
 temp = actual\_data[y]  
 actual\_data[y] = actual\_data[x]  
 actual\_data[x] = temp  
  
 return actual\_data

Rekombináció

Keresztezzük az egyedeket, méghozzá úgy, hogy meghatározunk 2 pontot, egy kisebbet, egy nagyobbat, majd nézzük a metszetét. Ezek mellett létrehozunk egy listát, amelybe az eredményünket pakoljuk.

Ha megfelelő részen vagyunk, akkor betesszük a listába.

Ha a 2-es listában lévő szám nem a metszet területén van, akkor azt a számot a listába tesszük.

A függvény azzal a listával tér vissza, amely a rekombinált adatokat tartalmazza.

def recombination(actual\_data1, actual\_data2, jobs):  
 first\_section = random.randint(0, int(jobs / 2) - 1)  
  
 second\_section = random.randint(int(jobs / 2) + 1, jobs - 1)  
  
 intersection = actual\_data1[first\_section:second\_section]  
  
 recombinated\_list = []  
  
 index = 0  
  
 for i in range(jobs):  
 if first\_section <= index < second\_section:  
 for k in intersection:  
 recombinated\_list.append(k)  
 index = second\_section  
 if actual\_data2[i] not in intersection:  
 recombinated\_list.append(actual\_data2[i])  
 index += 1  
 return recombinated\_list

Fitness

Folyamatosan számoljuk az eltelt időt, amíg van munkánk, majd ezt az értéket adjuk vissza. Méghozzá úgy, hogy előre meghatározunk 2 listát, a hátralévő és az elkészült munkáknak (gépenként).

Amíg az elkészült gépek száma nem egyenlő a munkák számával a következő történik: (itt mindig növeljük az időt)

Ha a jelenlegi gépnek még van dolga, akkor csökkentjük a jelenlegi munkát.

* El kell különítenünk az első gépet és a többi.
* Ha az aktuális gép végzett az aktuális munkával, akkor csökkentjük az értéket.

Ha a munkát elvégeztük, akkor növeljük a már elvégzett lista értékét.

* Biztosítjuk, hogy ne legyen végtelen loopunk
* Beállítjuk a következő munkát
* Ha az aktuális gép végzett az aktuális munkával, akkor csökkentjük az értéket.

def fitness(machines, jobs, job\_list, order\_of\_jobs):  
  
 currently\_used\_jobs = [-1 for \_ in range(machines)]  
 already\_done\_jobs = [0 for \_ in range(machines)]  
 for i in range(machines):  
 currently\_used\_jobs[i] = job\_list[order\_of\_jobs[0]][i]  
  
 time = -1  
  
 while already\_done\_jobs[machines - 1] != jobs:  
 time += 1  
 for i in range(machines):  
 if currently\_used\_jobs[i] != 0:  
 if i != 0:  
 if already\_done\_jobs[i] < already\_done\_jobs[i - 1]:  
 currently\_used\_jobs[i] -= 1  
 else:  
 currently\_used\_jobs[i] -= 1  
 else:  
 already\_done\_jobs[i] += 1  
 if jobs <= already\_done\_jobs[i]:  
 currently\_used\_jobs[i] = -1  
 already\_done\_jobs[i] = jobs  
 else:  
 currently\_used\_jobs[i] = job\_list[order\_of\_jobs[already\_done\_jobs[i]]][i]  
 if i != 0:  
 if already\_done\_jobs[i - 1] > already\_done\_jobs[i]:  
 currently\_used\_jobs[i] -= 1  
 else:  
 currently\_used\_jobs[i] -= 1  
  
 return time

Lista idő szerinti rendezése

def sort\_list(time\_base, list\_base, generations):  
 time = time\_base.copy()  
 \_list = list\_base.copy()  
 for i in range(generations - 1):  
 for k in range(i + 1, generations):  
 if time[i] > time[k]:  
 temp = time[i]  
 time[i] = time[k]  
 time[k] = temp  
 temp = \_list[i]  
 \_list[i] = \_list[k]  
 \_list[k] = temp  
 return time, \_list

Main program

best\_list tárolja a probléma legjobbnak ítélt megoldását

base\_temporary\_list az alaplista ideiglenes változata (másoláshoz szükséges)

best\_found\_time tárolja a probléma legjobban megtalált időpontját.

base\_list az alaplista, ezt legeneráljuk

Megismételjük a szimulációt minden egyes iterációnál.

Ellenőrizzük az újonnan talált időt, ha jobb, akkor felülírjuk a legjobb talált időt, és a legjobb listát, majd új alapot állítunk be a további generációkhoz.

Aztán kiíratjuk a megoldást.

def mainprogram(machines, jobs, max\_iterations, generations, job\_list):  
  
 best\_found\_time = float('inf')  
 base\_list = []  
  
 for i in range(jobs):  
 base\_list += [i]  
 best\_list = base\_list.copy()  
  
 file\_output = open("log.txt", "a")  
 file\_output.write("Base: " + str(base\_list) + "\n")  
 print("The genetic algorithm:")  
  
 for i in range(max\_iterations):  
 base\_temporary\_list, time = get\_new\_genetic(machines, jobs, job\_list,  
 base\_list, generations)  
  
 if best\_found\_time > time:  
 best\_found\_time = time  
 best\_list = base\_temporary\_list.copy()  
 base\_list = base\_temporary\_list.copy()  
  
 file\_output.write("Best found solution: " + str(best\_list) + "\nTime: " + str(best\_found\_time))  
 print("Best found solution: ", str(best\_list), "\nTime: ", best\_found\_time)  
 file\_output.close()

Main

Meghatározzuk a .csv file nevét, amiből olvasni szeretnénk majd.

Végig iterálunk a file-ból legenerált listákon, és minden iterációnál átadjuk az adatokat a main függvénynek.

A véletlenszám-generátornak szüksége van egy számra, amellyel elindulhat (egy magértékre), hogy véletlen számot tudjon generálni, így ezt megadjuk.

def main():  
 file\_name = 'base\_data.csv'  
 data\_list = filereader(file\_name)  
  
 for data in data\_list:  
 machines = int(data[0])  
 max\_iterations = int(data[1])  
 jobs = int(data[2])  
 generations = int(data[3])  
 seed\_of\_the\_generation = int(data[4])  
 random.seed(seed\_of\_the\_generation)  
  
 job\_list = randomizejobs(machines, jobs)  
  
 mainprogram(machines, jobs, max\_iterations, generations, job\_list)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()