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 olvasásához 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()
```