Mesterséges intetlligenciák

Féléves feladat

Készítette: **Kovács Krisztián**

Neptunkód: **WIQPM2**

**Probléma: Vehicle Routing Problem Tabu keresés algoritmussal.**

Az implementációt python nyelven készítettem.

Feladat megoldásának lépései:

1. Random függvény segítségével generáljuk a kocsik, illetve a pontok számát. Ezután a pontoknak generáljuk a koordinátáit.
2. Létrehozzuk a távolság mátrixot, mely egy négyzetes mátrix, mérete megegyezik a pontok számával. Kiszámoljuk az (i, j) pontok Manhattan távolságát.
3. Létrehozunk egy kezdő útvonalat, értékpárokból áll, (pont, futár) az alakja.
4. Definiálunk két függvényt: első a kocsinkénti távolságot számolja ki, amelyet lement egy tömbbe, a második az össztávolságot számolja ki a tömb elemeit összeadva.
5. A szomszédsági keresést definiáljuk, ami kétféle kell, hogy legyen. Az első részben egy futár útvonalát választja ki, és ott hajtjuk végre a cserét, a legjobb cserét követően le fogjuk menteni a jobb útvonalat és a pont-futár kombinációt lementjük a tabu listánkba. A második részben két random pontot választ és azoknak cseréli meg a kiszállítóját, a legjobb cserét követően le fogjuk menteni a jobb útvonalat és a pont-futár kombinációt lementjük a tabu listánkba. Visszatérünk a jobb megoldással.
6. Egy-egy függvényt definiálunk a konzolra való kiírásra és a fájlba való kiírásra.

Függvények és funkciójuk:

def distance(p1, p2) – két pontot kap és kiszámoljak a pontok közti távolságot

def dist\_initialize(loc) – a pontok számát kapja meg, egy tömbbel tér vissza, melybe a pontok két koordinátáját létrehozza

def vrp\_initialize(loc, veh) – a pontokhoz egy random futárt párosít és visszatér a kezdeti útvonallal

def transform\_vrp(vrp) – loc x loc mátrix a return-je, az (i, j) pontok közti távolságokat tárolja

def dstnc(dict, v, veh)- egy tömbbel tér vissza, melyben kocsinkként lebontva összeadja annak a megtett távolságát

def traveled(dist) – előzőleg említett tömböt kapja meg és visszatér a teljes megtett távval

def neighbor\_search(v, object\_f, veh, iterations, neighbor) – a legjobb útvonal megtalálására törekszik, futása végén visszaadja azt.

def print\_solution(v, veh, dist) – konzolra kiírja egyes kocsiknak az útvonalát, illetve ezeknek a megtett távolságát és össztávot is

def print\_file(loc, veh, vrp, v\_best, dist) – létrehoz egy fájlt, melybe kiírja pontok koordinátáit, a pontok és kocsik számát, ezeknek az útvonalát, megtett távolságát és össztávot is

A neighbor\_search működése:

Paraméterei: kezdő útvonal, a távolságot meghatározó függvény, kocsik száma, iterációk száma, illetve mennyi iterációból keresse meg a legjobb cserét.

Létrehozom a legjobb útvonalat, legjobb távolságot, ebbe ezeket fogom elmenteni, kezdeti értéke a kezdő útvonal és ennek a távolsága. Létrehozom a tabu listát, melynek méretét a lehetséges pont és kocsi kombináció felére állítottam. Egy iteráció \* szomszéd \* 2 hosszú ciklust hozok létre, két részre bontva. Az első részben kiválaszt egy kocsit és az útvonalában cseréket hajt végre és keresi a legjobb cserét, ha egy jobb cserét talál, akkor lementi és tabu listába teszi. A legjobb cserét lementi egy bázisba és kiszámolja a megtett utat. Ha ez a bázis jobb, akkor lementi a legjobb útvonalba, és a bázisban folytatja majd a következő cseréket, ha nem jobb, akkor is ezt a bázist fogja felhasználni a következő cseréknél, viszont mentés nélkül. A második rész is hasonlóan működik, viszont ott nem egy futár útját tekinti, hanem kiválaszt két random pontot, melyre különböző kocsik mennek, és ott hajt végre cseréket. A végén visszatér a talált legjobb útvonallal.

Úgy van megoldva, hogy a futárok megegyező számú pont mennyiséget látogatnak, annak érdekében, hogy minél hamarabb legyen kiszállítva a csomag.

A képen szöveg látható

Automatikusan generált leírás

A képen szöveg látható

Automatikusan generált leírás