Problema comis-voiajorului

Berar George 30641

Descrierea problemei

Se considera n orase si un comis-voiajor care trebuie sa viziteze toate orasele, trecand o singura data prin fiecare si sa se intoarca in orasul initial astfel incat costul total sa fie cat mai mic. Solutia poate fi reprezentata ca o permutare de n orase. Costul unei solutii este suma distantelor dintre orase in ordinea in care apar in permutare.

Prezentarea algoritmilor

1. **Metoda exhaustiva**

Aceasta metoda reprezinta o abordare directa a problemei in care se incearca toate permutarile posibile din spatiul de cautare pentru a gasi cea mai buna solutie. Deoarece spatiul de cautare este determinat de numarul de orase prin care trebuie sa treaca comis-voiajorul, acesta poate sa fie extrem de mare daca si numarul de orase este mare. Pentru un numar de 51 de orase, spatiul de cautare este format din 51! de permutari. Metoda exhaustiva genereaza toate permutarile posibile cu cele n orase dupa care determina care este ruta cu costul cel mai mic.

Desi se obtine solutia cea mai buna, faptul ca spatiul de cautare este extrem de mare implica nevoia de resurse superioare pentru ca timpul de calcul sa fie unul redus. Aceasta abordare directa este perfecta pentru cazul in care avem un numar redus de orase, de preferat sub 10.

**ALGORITHM exhaustiveSearch(TSPInstance)**

**permutations[] = generateAllPermutations(TSPInstance)**

**bestCost = INITIAL\_VALUE**

**bestRoute = [1..N]**

**for i to nrOfPermutations do**

**cost = computeCost(permutations[i])**

**if(cost < bestCost) then**

**bestCost = cost**

**bestRoute = permutations[i]**

1. **Metoda Local Search**

Aceasta metoda porneste cu o solutie aleatoare din spatiul de cautare dupa care incearca sa o imbunatateasca. Solutia aleatoare reprezinta o permutare aleatoare a oraselor. Pasii algoritmului sunt urmatorii:

**1** se alege o solutie aleatoare din spatiul de cautare; aceasta reprezinta solutia curenta

**2** se aplica o transformare asupra acestei solutii pentru a obtine o solutie noua; pentru transformare am folosit 2-opt\*

**3** daca noua solutie este mai buna decat solutia curenta, adica costul este mai mic, atunci noua solutie devine solutia curenta

**4** se repeta pasii 2 si 3 pana cand nu se mai poate imbunatatii solutia

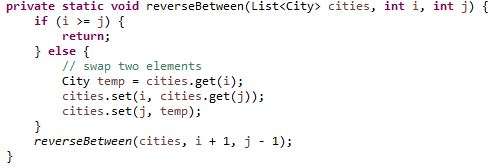
**\*2-opt**

Se aleg aleator doi indici, i si j, astfel incat 1 <= i < j <= n, unde n reprezinta numarul de orase. Conditia necesara o reprezinta ca distanta dintre i si j sa fie minim 3. Dupa ce au fost alesi, in solutie se inverseaza orasele dintre acesti 2 indici.

Ex. Solutia curenta 1 2 ***3 4 5*** 6, i = 1, j = 5

Dupa aplicarea transformarii 2-opt => 1 2 ***5 4 3***6

Mai jos este prezentata o implementare in Java a metodei de inversare intre cei doi indici:



1. **Metoda Greedy**

Aceasta metoda se bazeaza pe diferite euristici pentru a alege cea mai buna ruta. Euristica pe care am folosit-o poarta numele de **nearest neighbour\*,** si se bazeaza pe alegerea, in orice moment, a celui mai apropiat vecin fata de punctul curent.

\* Incepand de la un oras oarecare, se alege cel mai apropiat oras nevizitat inca si se continua procesul pana cand toate orasele au fost vizitate (de la ultimul oras ne intoarcem in primul pentru o ruta completa).

**ALGORITHM greedySearch(TSPInstance)**

**route[] = [1..N]**

**startingCity = TSPInstance.getCity(0)**

**route.add(startingCity)**

**startingCity.setVisited(true)**

**while( notAllCitiesAreVisited ) do**

**neighbour = getNearestNeighbour(TSPInstance.getNevisitedCities(), startingCity)**

**route.add(neighbour)**

**neighbour.setVisited(true)**

**startingCity = neighbour**

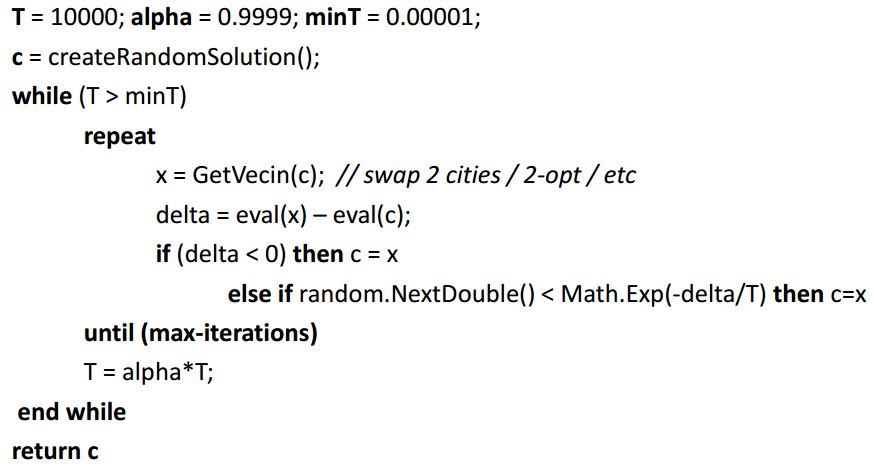
O abordare mai optimizata ar fi sa se gaseasca cea mai buna ruta, pornind din fiecare oras, iar apoi din toate rutele sa se aleaga ruta cu costul cel mai mic. Aplicand aceasta optimizare pe algoritmul de mai sus, am obtinut o imbunatatire semnificativa. Ex. Pentru 51 de orase:

1. Abordare neoptimizata -> cost = 511
2. Abordare optimizata -> cost = 482

Pasii noului algoritm:

1. Selecteaza oras de start
2. Gaseste ruta completa si costul
3. Repeta 1 si 2 pana cand nu se mai poate selecta un oras de start
4. Alege cea mai buna ruta
5. **Metoda Simulated-Annealing**

Aceasta metoda se bazeaza pe folosirea unui parametru, numit temperatura, pentru a schimba probabilitatea de a ne muta dintr-un punct in altul al spatiului de cautare. Metodele prezentate mai sus, tind sa dureze un timp indelungat pentru a calcula solutia, exemplu cel exhaustiv, sau obtin solutii aproximative care ar putea insemna un optim local. Simulated-Annealing ne ajuta prin folosirea acestui parametru sa iesim dintr-un optim local.



Modificarea parametrului T, numit temperatura, influenteaza probabilitatea cu care alegem urmatorul punct(permutare). Cu cat T este mai mic cu atat este mai putin importanta evaluarea punctelor.

Rezultate experimentale

