10.11.2012

Obligatorisk oppgave 1

av

Sturla Gunnerød, Vegard Knudsen og Kristian Masdal

Traveling Salesman Problem - TSP
(https://www.localsolver.com/docs/last/_images/tsp.png)

Innholdsfortegnelse

Innledning

Denne oppgaven tar for seg det kjente problemet “Traveling salesman problem”, og prøver å løse det med ulike algoritmer. Tre algoritmer ble implementert: “random algorithm”, “iterativ random algorithm”, og “greedy algorithm”. Disse algoritmene vil generer en initiell løsning. Deretter ble det implementert to nye algoritmer som hver for seg tar inn én initiell løsning og prøver å optimalisere løsningen. Til slutt utfordres algoritmene med forskjellige størrelser, og resultater sammenlignes.

Problemstilling/oppgavetekst

Compulsory Assignment 1

In this assignment , different algorithms are going to be tested for solving the traveling salesman
problem (TSP).

1: Implement random algorithm, iterative random algorithm and greedy algorithm in order to obtain an initial solution.
2: Proceed with two optimization algorithms in order to optimize the solutions obtained in question
    1. (greedy optimization, greedy-Random optimization algorithms)
3. Compare the algorithms implemented in 1.
4. Compare the global algorithms (1+2) .
5. Use problems with different sizes: 1000, 5000, 10000.

Fremgangsmåte/Pseudo kode

All kode ble skrevet i programmeringsspråket Java.

Traveling Salesman Problem - TSP

Først ble TSP implementert ved hjelp av en todimensjonal matrise, hvor x-retning representerte forskjellige byer og y-retning representerer dens naboer. Tallet i denne posisjonen er prisen det koster å dra fra by x til by y. Prisen er trukket tilfeldig fra et tall mellom 1 og 100. Ved å sette diagonalen til matrisen lik null, eliminerer vi at byen er nabo til seg selv. (I denne oppgaven har vi gjort slik at alle byer er naboer med alle byer utenom seg selv).

Random Algorithm

Det ligger i navnet “Random algorithm”, algoritmen velger tilfeldige byer. Det er et par ting man bør være observant på: at man ikke trekker sin egen posisjon, at byen ikke er besøkt, og at verdien ikke er lik 0 i denne posisjonen. Hvis noen av disse problemene forekommer må vi trekke en ny by å dra til. Skulle det gå bra vil en “besøkte Byer-matrise” oppdateres og vi flytter oss videre. Algoritmen iterer helt til alle byer er besøkt.

Pseudo kode:

```
a = tilfeldig by
while(teller ikke er lik by-lengde){
        do{
                b = trekker tilfeldig by å dra til;
        }while( egen posisjon || besøkt || posisjonens verdi = 0)
        besøkte byer oppdateres;
        pris += kostnaden fra by a til by b;
        teller++;
        a = b; // beveger oss til neste by
}
return pris, løsning;
```

Big O notasjon = O(n^2)

Iterativ Random Algorithm

Denne algoritmen tar for seg forrige algoritme “Random algorithm”, og iterer denne algoritmen flere ganger for å finne en bedre tilfeldig løsning. Mengden itereringer ble satt til 100 ganger. Det er viktig i denne algoritmen at vi tar vare på løsningene “beste løsning” og “nåværende løsning” slik at vi kan sammenligne de for hver iterasjon.

Pseudo kode:

```
for(teller; teller < iterasjoner; teller++){
```

```
        nåværende rute = random algorithm();
        if(nåværende rute er bedre enn beste rute){
                beste rute = nåværende rute;
        }
}
return beste rute;
```

Big O notasjon = O(n^3)

Greedy Algorithm

Den grådige algoritmen velger den laveste kostnaden det koster å dra til en nabo. Den vil derfor trenge å iterere gjennom alle byene før den bestemmer seg hvor den skal dra til. Man er nødt til å sammenligne underveis og bør være observant på at man får tak i den beste verdien, at denne posisjonen ikke er besøkt og at verdien ikke er lik 0.

Pseudo kode:

```
for(teller; teller < by-lengde; teller++){
        beste Verdi = maks størrelse;
        for(teller; teller < by-lengde; teller++){
                if(nåværende verdi < beste verdi && ikke besøkt && ikke er null){
                        beste verdi = nåværende verdi;
                }
        }
        besøkte byer oppdateres;
        pris += kostnaden fra by a til by b;
        flytter oss til beste by;
}
return pris, løsning;
```

Big O notasjon = O(n^2)

Greedy optimization Algorithm

Denne algoritmen tar for seg en initiell løsning fra en av de tre første algoritmene. Den bytter plasser på to tilfeldige byer fra det initielle problemet, og løser deretter problemet. Den nye løsningen vil sammenlignes med den initielle løsningen. Algoritmen prøver dette til x-antall ganger.

Pseudo kode:

```
for(teller; teller < x-antall-ganger; teller++){
        do{
                a = tilfeldig by;
                b = tilfeldig by;
        }while( a er lik b);
        for(teller; teller < by-lengde; teller++){
                byene byttes om
        }
```

```
        ny kostnad = en av de initielle algoritmene( med byttet om byer);
                if(ny kostnad er bedre enn gammel kostnad ){
                        gammel kostnad = ny kostnad;
                        gammel løsning = ny løsning
                        if(ny kostnad er bedre enn beste kostnad){
                                beste kostnad = ny kostnad;
                                best løsning = ny løsning;
                        }
}
return beste kostnad, ny løsning;
```

Big O notasjon = O(n^2) + den initielle algoritmen

Greedy Random Optimization Algorithm

Denne optimaliserings algoritmen er veldig lik den forrige. Det tilfeldige elementet i denne algoritmen kommer av at man av og til aksepterer en dårligere løsning for å kunne utvide søkeparametrene. Algoritmen vil statistisk sett akseptere dårligere løsninger tidlig i kjøringen, men synke ettersom algoritmen har iterert flere ganger og vi nærmer oss den beste løsningen.

Pseudo kode:

```
do{
        for(teller; teller < x-antall-ganger; teller++){
                do{
                        a = tilfeldig by;
                        b = tilfeldig by;
                }while( a er lik b);
                for(teller; teller < by-lengde; teller++){
                        byene byttes om
                }
                if(ny kostnad er bedre enn gammel kostnad ){
                        gammel kostnad = ny kostnad;
                        gammel løsning = ny løsning
                        if(ny kostnad er bedre enn beste kostnad){
                                beste kostnad = ny kostnad;
                                best løsning = ny løsning;
                        }
                else{
                        x = tilfeldig tall mellom 0.00 og 0.99;
                        if(x < sannsynlighet){
                                gammel kostnad = ny kostnad;
                                gammel løsning = ny løsning;
                        }
                }
        sannsynlighet = sannsynlighet * 0.9;
}while(sannsynlighet > 0.0000001)
```

return beste kostnad, beste løsning
Big O notasjon = O(n^2) + den initielle algoritmen

Resultater

Her er resultater fra kjøringer med forskjellige størrelse på antall byer. Kostnad mellom 1 og 100.

Kjøring 1:

```
Størrelse: 1000
Random resultat: 52153
Greedy optimalisert på random: 49595
Greedy random optimalisert på random: 46927

Iterativ Random resultat: 48443
Greedy optimalisert på iterativ random: 47225
Greedy random optimalisert på iterativ random: 46317

Greedy resultat: 1005
Greedy optimalisert på greedy: 1002
Greedy random optimalisert på greedy: 1000
```

Kjøring 2:

```
Størrelse: 5000
Random resultat: 252649
Greedy optimalisert på random: 250056
Greedy random optimalisert på random: 245942

Iterativ Random resultat: 246052
Greedy optimalisert på iterativ random: 244117
Greedy random optimalisert på iterativ random: 242306

Greedy resultat: 4999
Greedy optimalisert på greedy: 4999
Greedy random optimalisert på greedy: 4999
```

Kjøring 3:

```
Størrelse: 10000
Random resultat: 501791
Greedy optimalisert på random: 499119
Greedy random optimalisert på random: 494352

Iterativ Random resultat: 498869
Greedy optimalisert på iterativ random: 496734
Greedy random optimalisert på iterativ random: 491081

Greedy resultat: 9999
Greedy optimalisert på greedy: 9999
Greedy random optimalisert på greedy: 9999
```

Konklusjon

Av de tre initiell løsningene gir selvfølgelig greedy best resultat og løsning med tanke på pris. De to algoritmene “random algorithm” og “iterativ random algorithm” gir løsninger som er veldig nært **Størrelse * (Kostnad/0.5).** Dette er ikke så rart da algoritmen i gjennomsnitt trekker halvparten av kostnad som pris.

De to optimaliseringsalgoritmene forbedrer kun resultat med et par prosent på “random algorithm” og “iterativ random algorithm”. “Greedy algorithm” får minimal til ingen forbedring da den allerede på første iterasjon har funnet den beste løsningen