Univerza v Ljubljani
Fakulteta za matematiko in fiziko

Žan Jernejčič in Ines Šilc

Reševanje problema trgovskega potnika s k-optimalnim in Lin-Kernighanovim algoritmom

Ljubljana, 2020

1 Definiranje problema

V projektni nalogi bova reševala Problem trgovskega potnika s pomočjo k-optimalnega in Lin-Kernighanovega algoritma.

Problem trgovskega potnika oziroma Travelling salesman problem (krajše TSP) je problem, kjer imamo podanih n mest in razdalje med vsemi (za vsak par mest imamo torej podano, koliko sta si oddaljeni). Zanima nas, ali lahko obiščemo vsako mesto in se na koncu vrnemo v prvotno mesto. Če označimo $d_{i,j}$ kot razdaljo mad i-tim in j-tim mestom, iščemo torej:

$$\min_{\pi \in S_n} \sum_{i=1}^{n-1} d_{\pi(i),\pi(i+1)} + d_{\pi(n),\pi(1)}$$

kjer je S_n množica vseh permutacij danih n mest.

Naivna rešitev je očitna, pogledamo $(n-1)!$ kombinacij, torej iz vsakega mesta v vsako drugo mesto, si zapišemo vse kombinacije in kakšno razdaljo smo prepotovali, ter izberemo tisto možnost, kjer je bila razdalja najkrajša.

Pred začetkom se lahko vprašamo kakšen mora biti graf, da lahko na njem izvajamo sledeča algoritma. Graf **ne sme** imeti **negativnih ciklov**, saj je najcenejši cikel potem očiten, in ustvarimo neskončno zanko, saj bo imela najboljša rešitev ceno -∞. Ker imamo opravka s cikli, lahko cikel začnemo v kateremkoli vozlišču, zato lahko vedno gledamo možne poti od začetka.

Za projekt sva uporabila programski jezik `python`. Uporabljala sva naslednje pakete:

- `networkx`: za definiranje in generiranje grafov
-
-
- `random`: za generiranje naključnih števil in seznamov
- `matplotlib`: za izrisovanje grafov
- `time`: za mertive časovne zahtevnosti

Za preverjanje veljavnosti poti v grafu za problem potujočega trgovca, morava vedeti:

- Vsaka pot trgovca bo morala imeti vsa vozlišča zato vedno preverimo:

  ```
  pot.order() == originalen_graf.order()
  ```

- Vsaka pot trgovca bo morala imeti točno toliko povezav kot vozlišč:

  ```
  pot.size(weight = None) == pot.order()
  ```

- Dolžina poti: `pot.size()`

2 k-optimalni algoritmi

2.1 2-opt algoritem

2.2 3-opt algoritem

3 Lin-Kernighanov algoritem

4 Viri

1. A. Hagberg, D. Schult, P. Swart:*NetworkX Refrence, Release 2.4*, [ogled 2. 1. 2020], dostopno na `https://networkx.github.io/documentation/stable/_downloads/networkx_reference.pdf`