

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 med 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 očitno, in ustvarimo neskončno zanko, saj bo imela najboljša rešitev ceno $-\infty$. 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 merjenje časovne zahtevnosti
- `itertools`: za generiranje prvotne permutacije

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

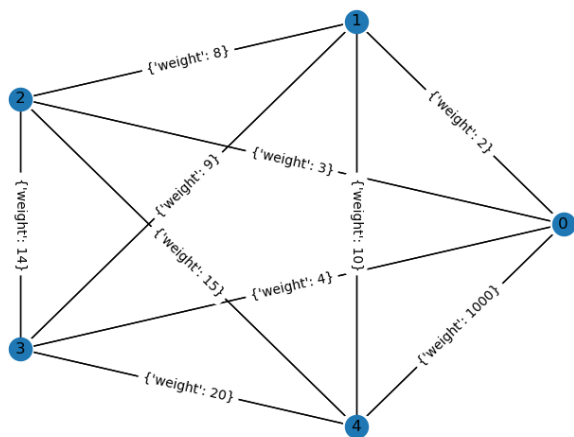
K-optimalni algoritem, je lokalni algoritem, kjer iščemo rešitev Problema trgovskega potnika in sorodnih problemov. Algoritem deluje tako, da zmanjša dolžino trenutnega potovanja, dokler ne dosežemo poti, katere dolžine ne moremo izboljšati.

Poznamo dve glavni različici k-opt algoritma, to sta 2-Opt in 3-Opt algoritem. 2-Opt algoritem deluje tako, da odstrani dve vozlišči grafa (dve mesti), tako razdeli pot na dva dela, poišče najboljšo rešitev podproblema, in nato poveže nazaj mesti na najboljši možen način. 3-Opt algoritem odstrani tri povezave ali vozlišča, tako dobimo 3 podomrežja mest. V naslednjem koraku analiziramo najboljšo pot med temi tremi podomrežji. To ponavljamo za druge tri povezave, dokler nismo poskusili vseh možnih poti v danem omrežju.

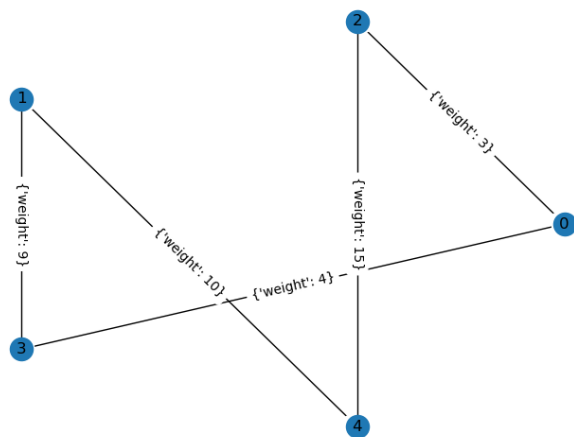
2.1 2-opt algoritem

```
def dva_opt(graf, pot):  
  
    najboljsa_pot = pot  
  
    izboljsanje = True  
    while izboljsanje:  
        izboljsanje = False  
        for i in range(1, len(pot) - 2):  
  
            for j in range(i + 1, len(pot)):  
  
                if j - i == 1: continue  
                nova_pot = pot  
                nova_pot[i:j] = pot[j - 1:i - 1:-1]  
  
                if cena_poti(graf, nova_pot) < cena_poti(graf, najboljsa_pot):  
                    najboljsa_pot = nova_pot  
                    izboljsanje = True  
    pot = najboljsa_pot  
  
    return (pot, cena_poti(graf, pot))
```

2.2 3-opt algoritem



Slika 1: Primer



Slika 2: Rešitev

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