Rješavanje problema trgovačkog putnika upotrebom pohlepnog algoritma

Bubanj, Deana

Sveučilište J. J. Strossmayera u Osijeku

Odjel za matematiku

Sveučilišni preddiplomski studij Matematika i računarstvo

Deana Bubanj

Rješavanje problema trgovačkog putnika upotrebom pohlepnog algoritma

Završni rad

Osijek, 2022

Sveučilište J. J. Strossmayera u Osijeku

Odjel za matematiku

Sveučilišni preddiplomski studij Matematika i računarstvo

Deana Bubanj

Rješavanje problema trgovačkog putnika upotrebom pohlepnog algoritma

Završni rad

Mentor:

doc. dr. sc. Domagoj Ševerdija

Komentori:

dr. sc. Mateja Đumić
dr. sc. Rebeka Čorić

Osijek, 2022

Sažetak

Problem trgovačkog putnika je problem kombinatorne optimizacije kojem je cilj pronaći najkraću rutu u kojoj će se obići svi dani gradovi i vratiti u početni iz kojeg se krenulo. U problemu promatranom u ovom završnom radu, gradovi su zadani preko koordinata, a bridovi koji spajaju dva grada nose unaprijed zadanu težinu. Cilj je da suma težina prijeđenog puta bude minimalna. Problem su prvi put opisali matematičari William Rowan Hamilton i Thomas Kirkman 1800-ih godina. Napravili su igru *Icosian game* koja se mogla riješiti korištenjem Hamiltonovog ciklusa. Igrač ima na raspolaganju dvadeset točaka i cilj je da pronađe Hamiltonov ciklus tako da svaku točku posjeti jednom i početna točka bude jednaka završnoj. U ovom radu je implementiran pohlepni pristup za rješavanje problema trgovačkog putnika i prikazano je napravljeno grafičko sučelje.

Ključne riječi

problem trgovačkog putnika, Hamiltonov ciklus, pohlepni algoritam

Solving the travelling salesman problem using a greedy algorithm

Abstract

The Travelling Salesman Problem is a problem task whose objective is to find the shortest route that includes all given cities and returns to the starting point. The cities of a route are given as coordinates and the edges connecting two cities have a certain weight. The goal is to keep travel costs as low as possible. The problem was first described by mathematicians William Rowan Hamilton and Thomas Kirkman in the 1800s. They created an Icosian game that was solved by finding a Hamiltonian cycle. The player has twenty points available and the goal is to find a Hamiltonian cycle where he visits each point once and the end point is the same as the starting point. In this paper, a greedy approach to solving this problem is implemented and a graphical interface is provided.

Key words

traveling salesman problem, Hamiltonian cycle, greedy algorithm

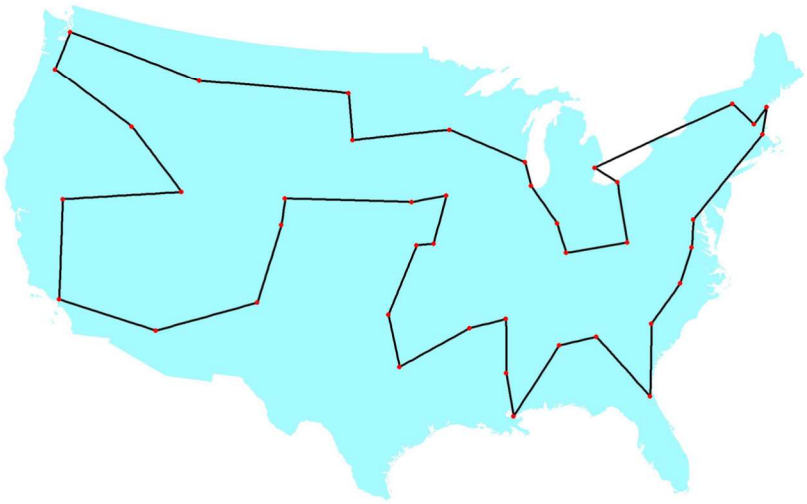
Sadržaj

1 | Uvod

Problem trgovačkog putnika ili skraćeno TSP (engl. *Travelling Salesman Problem*) je jedan od poznatijih problema kombinatorne optimizacije. Cilj ovog problema je pronaći najkraću rutu koja će obići sve zadane gradove i vratiti se u početni grad. Gradovi su predstavljeni vrhovima unutar grafa te su međusobno povezani s bridovima kojima je poznata težina. U literaturi ovaj problem je moguće pronaći u brojnim radovima, a razvojem tehnologije dolazi do razvoja sve uspješnijih metoda za rješavanje ovog problema. Prvi put ga uvode matematičari William Rowan Hamilton i Thomas Kirkman 1800-ih godina. Napravili su igru *Icosian game* koja se rješava pronalaskom Hamiltonovog ciklusa [5]. U ovom radu naglasak je na rješavanju simetričnog problema trgovačkog putnika, što bi značilo da je udaljenost između dvije točke jednaka u oba smjera. Za rješavanje simetričnog problema trgovačkog putnika u ovom radu koristit će se pohlepni algoritam. Pohlepni algoritam u svakom koraku u rutu dodaje grad koji u tom trenutku daje najbolje rješenje i tako inkrementalno rješava problem. Poglavlje 2 opisuje promatrani problem, grafove i metrike, poglavlje 3 prikazuje pseudokod algoritma, te manji primjer, dok se poglavlje 4 bavi detaljima implementacije problema u programskom jeziku Python, te vizualizacijom u grafičkom sučelju Pygame.

2 | Opis problema

Problem trgovačkog putnika logistički je problem na koji nailazimo u svakodnevnom životu. Trgovački putnik kreće iz jednog grada i cilj je da u što kraćem vremenu obiđe zadane gradove te se vrati u početni grad. U rješavanju se uglavnom koriste grafovi. Svaki grad označava vrh grafa, početni grad je početni vrh/točka, dok su bridovi grafa putovi između dva vrha koji sadrže neku informaciju (trošak/težinu). Na primjene tog problema nailazimo kod planiranja rasporeda vožnje autobusa, vozila za čišćenje snijega i sl. Problem trgovačkog putnika prvi put su riješili Dantzig, Fulkerson i Johnson 1954.g. na stvarnom primjeru 49 gradova Sjedinjenih Američkih država [2]. Primjer je prikazan na slici 2.1.



Slika 2.1: Problem trgovačkog putnika s 49 gradova.

2.1 Neusmjeren graf

Definicija 1. *Graf je uređena trojka $G = (V, E, \phi)$, gdje je $V = V(G)$ neprazan skup čije elemente nazivamo vrhovima, $E = E(G)$ je skup disjunktan s V čije elemente nazivamo bridovima i ϕ je funkcija koja svakom bridu e iz E pridružuje par (u, v), ne nužno različitih vrhova $u, v \in V$. Graf skraćeno označavamo $G = (V, E)$ ili samo G.*

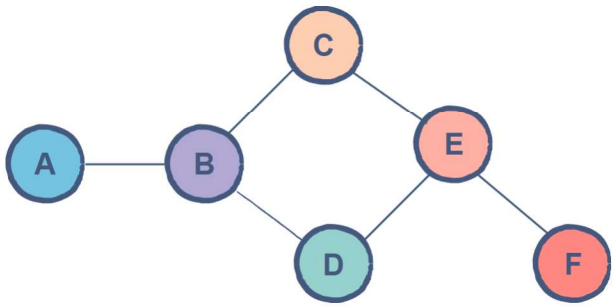
U ovom radu graf je osnovna struktura koju koristimo u implementaciji problema. Najvažnija funkcija u klasi graf je $isCyclic()$ koja provjerava imamo li ciklus u grafu. U daljnjem tekstu bit će objašnjena i njezina uloga u implementaciji algoritma.

```
def isCyclic(self):

    visited = [False]*(self.V)
    for i in range(self.V):
        if visited[i] == False:
            if(self.isCyclicUtil
                (i, visited, -1)) == True:
                return True

    return False
```

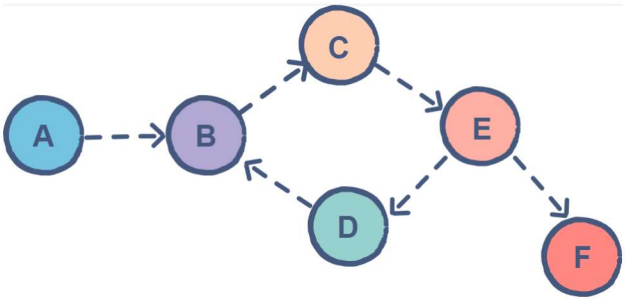
Program 2.1: Primjer funkcije isCyclic()

Ovo je rekurzivna funkcija koja se poziva na grafu. Vraća True ili False u ovisnosti imamo li ciklus u grafu ili ne.

Na slikama 2.2 i 2.3 vidimo razliku usmjerenog i neusmjerenog grafa. Oba grafa imaju ciklus. U implementaciji koja će biti objašnjena u daljnjem tekstu to se ne smije dogoditi. Kao što je na početku rečeno, inačica TSP-a promatrana u ovom radu je simetričan problem i iz tog razloga možemo koristiti neusmjeren graf [7].



Slika 2.2: Primjer neusmjerenog grafa.



Slika 2.3: Primjer usmjerenog grafa.

2.2 Metrike

Kod pokretanja algoritma možemo birati koju metriku želimo koristiti u računanju. Implementirane su Euklidska i Manhattan metrika. U ovisnosti koju metriku izaberemo krajnji rezultat bit će drugačiji.

2.2.1 Euklidska metrika

Formulu za udaljenost prvi put je objavio Alexis Clairaut 1731.g., iako ju je on objavio ime je dobila po grčkom matematičaru Euklidu koji se prvi bavio problemom računanja udaljenosti [8].

Definicija 2. *Neka je $\|\cdot\|$ euklidska norma na R^n. Preslikavanje $d : R^n \times R^n \to R$ zadano formulom $d(x,x) := \|x - y\|$ zove se euklidska udaljenost ili euklidska metrika na skupu R^n.*

Euklidsku udaljenost zovemo još i L2-metrikom [1]. Udaljenost između dvije točke u 2-dimenzionalnom prostoru dana je sa:

$$d_2((x_1, y_1), (x_2, y_2)) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}.$$

Na slici 2.4 je vidljiva implementacija L2 udaljenosti korištene u radu iz koje se vidi da su u radu korištene točke sa samo 2 koordinate.

```
xd = x[i] - x[j];
yd = y[i] - y[j];
dij = nint( sqrt( xd*xd + yd*yd) )
```

Slika 2.4: L2 udaljenost korištena u implementaciji.

2.2.2 Manhattan metrika

Manhattan metrika dobila je ime po gradskoj četvrti Manhattan. Vrhovi grafa u ovom slučaju su raskrižja, a bridovi ulice jer je gradska četvrt u obliku mreže. Najkraća udaljenost između dva raskrižja računata euklidskom metrikom zahtijeva prolazak kroz blokove grada što je sporije nego kretanje od jednog do drugog raskrižja koristeći Manhattan metriku [4].

Manhattan metriku zovemo još i L1-metrikom [1]. Manhattan udaljenost između dvije točke u 2-dimenzionalnom prostoru dana je sa:

$$d_1((x_1, y_1), (x_2, y_2)) = |x_2 - x_1| + |y_2 - y_1|.$$

Na slici 2.5 prikazana je implementacija L1 udaljenosti koja je korištena u kodu za računanje udaljenosti točaka u 2-dimenzionalnom prostoru.

```
xd = abs( x[i] - x[j] );
yd = abs( y[i] - y[j] );
dij = nint( xd + yd );
```

Slika 2.5: L1 udaljenost korištena u implementaciji.

3 | Pohlepni algoritam

Pohlepni algoritam (engl. *greedy algorithm*) jest pristup za rješavanje problema odabirom najbolje dostupne opcije u trenutku odluke. Zbog toga, algoritam nikad ne poništava radnju, tj. nakon odluke više ne provjera je li odluka bila ispravna ili se nekom drugom odlukom moglo doći do boljeg rješenja. Algoritam neće uvijek dati najbolje rješenje za sve probleme. Pohlepnim algoritmom je moguće rješavati probleme optimizacije neovisno radi li se o minimizaciji ili maksimizaciji. Uspješno se primjenjuje na pronalaženju grafa minimalnog razapinjujućeg stabla (Primov ili Kruskalov algoritam) te pronalaženje najkraćeg puta između dva vrha (Dijsktrin algoritam) [3]. Jedna od prednosti pohlepnog algoritma je njegova laka implementacija.

3.1 Pseudokod pohlepnog algoritma

Algoritam 1 Pohlepni algoritam

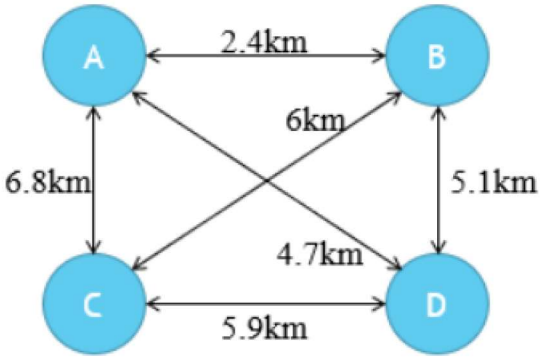
Ulaz: graf s vrhovima i bridovima
Izlaz: ruta

1: Sortiraj bridove po težinama u rastućem poretku
2: **Ako** brid nije posjećen i stupanj vrha na koji dodajemo brid neće biti 3 i ne tvori ciklus veličine manje od N **onda**
3: Dodaj brid u rutu
4: **Ako** imamo manje od N bridova u ruti **onda**
5: Vrati se na korak 2
6: **Inače**
7: Vrati novu rutu

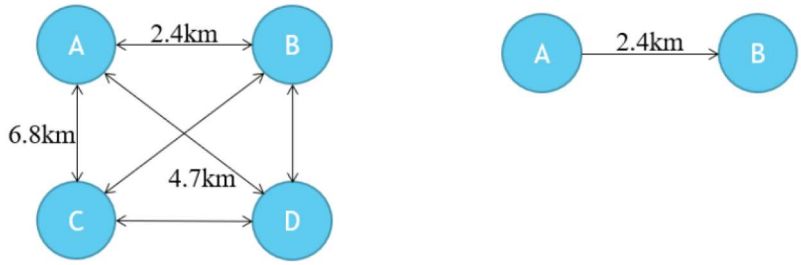
Algoritmom 1 dan je pseudokod pohlepnog algoritma za rješavanje problema trgovačkog putnika. Kao ulaz ovaj algoritam prima graf s njegovim vrhovima i bridovima. Vrhovi grafa su gradovi kojih ima N. Iz pseudokoda algoritma je vidljivo da se u svakoj iteraciji dodaje po jedan brid dok se u rutu ne dodaju svi gradovi. Prilikom odabira brida potrebno je provjeriti određene uvjete koji su dani u koraku 2.

3.2 Prikaz pohlepnog algoritma na manjem primjeru

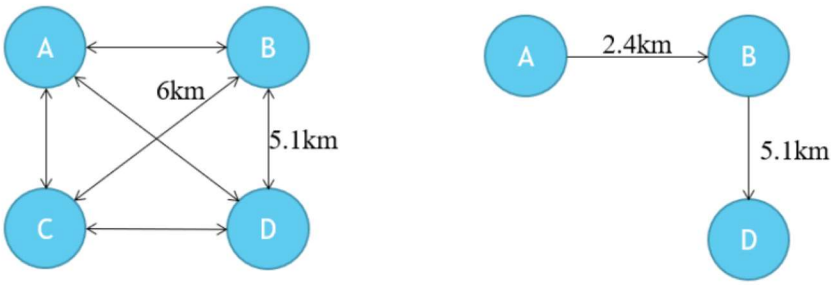
U ovom potpoglavlju na jednom manjem problemu prikazat će se kako radi pohlepni algoritam. Graf se sastoji od 4 čvora i 6 bridova. Na svakom bridu je naznačena udaljenost između dva povezana čvora. Slika 3.1 prikazuje zadani problem na koji će biti primijenjen pohlepni algoritam, dok slika 3.2 prikazuje korake opisanog algoritma. Ovim postupkom dobit ćemo listu vrhova redom kojim su spajani $A \to B \to D \to C \to A$ i ukupni trošak $2.4 + 5.1 + 5.9 + 6.8 = 20.2$.



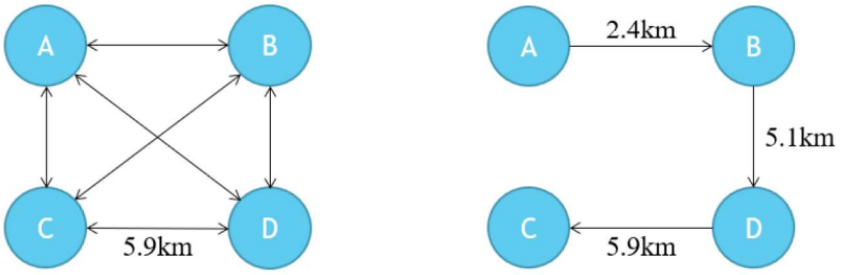
Slika 3.1: Zadani vrhovi i bridovi.



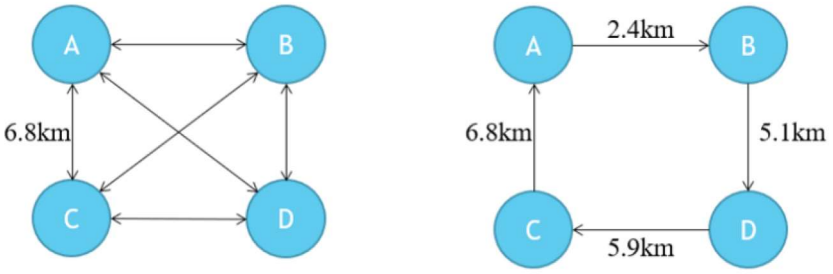
(a) Prvi korak.



(b) Drugi korak.



(c) Treći korak.



(d) Četvrti korak.

Slika 3.2: Pronalazak najmanje udaljenost.

4 | Implementacija i vizualizacija

Implementacija praktičnog dijela ovog rada napravljena je u programskom jeziku Python, dok je za vizualizaciju korišteno grafičko sučelje Pygame. Instance problema dolaze iz skupa problema TSPLib [6] i zadane su u obliku prikazanom na slici 4.1. Prva i druga točka označavaju vrhove koji se dodaju u graf, dok treća točka odgovara težini brida između dva zadana vrha.

```
1 41 49
2 35 17
3 55 45
4 55 20
5 15 30
6 25 30
```

Slika 4.1: Prikaz nekoliko točaka iz instance problema trgovačkog putnika.

4.1 Implementacija

```
def GreedyAlgorithm(weights, graph_len):
    visited_nodes_and_final_cost = []
    list_of_visited_nodes2 = []
    final_cost2 = []
    g = Graph(graph_len)
    sorted_weights = {k: v for k, v in sorted(weights.items(),
    key=lambda item: item[1])}
    for key, value in sorted_weights.items():
        if (key[0] not in list_of_visited_nodes2 or key[1] not in
     list_of_visited_nodes2) and((list_of_visited_nodes2.count(
    key[0])<2) and (list_of_visited_nodes2.count(key[1]) <2)):
            list_of_visited_nodes2.append(key[0])
            list_of_visited_nodes2.append(key[1])
            final_cost2.append(value)
            g.addEdge(key[0]-1, key[1]-1)
    for i in range(0, len(list_of_visited_nodes2), 2):
        del sorted_weights[(list_of_visited_nodes2[i],
    list_of_visited_nodes2[i+1])]
```

Program 4.1: Prvi dio koda.

Na početku definiramo tri liste potrebne za spremanje podataka: listu posjećenih čvorova, završnog troška i listu koja će sadržavati prethodno navedene dvije. Uz liste definiramo

i graf g. Početne rute zadane su u obliku rječnika. Ključevi u rječniku su parovi točaka, dok je vrijednost težina koju nosi brid između dvije točke. Kako je na početku rečeno cilj algoritma je da u svakom koraku traži najbolje moguće rješenje, zbog toga sve težine koje funkcija primi na početku sortiramo lambda funkcijom od najmanje do najveće. Lista posjećenih čvorova nam služi kako bismo u nju spremali sve čvorove koje smo do sada posjetili. Na početku prolazimo rječnikom sortiranih težina i redom čvorove stavljamo u listu posjećenih čvorova, dok u listu final cost dodajemo vrijednosti iz rječnika. Ukupnu težinu dobijemo zbrajanjem svih vrijednosti u listi final cost. U prethodno definirani graf redom dodajemo čvorove. Trenutno ne trebamo provjeravati ima li graf ciklus jer kod dodavanja se dodaju samo čvorovi koji do sad nisu posjećeni. U ovom prolasku algoritam će spojiti prvo sve susjedne točke. Krajnji rezultat dan je kao na slici 4.2. Na kraju prolaskom kroz rječnik sortiranih težina brišemo sve one čvorove koje smo spojili kako ne bismo opet njih provjeravali u daljnjem kodu.

```
for key, value in sorted_weights.items():
    g.addEdge(key[0]-1,key[1]-1)
    if(g.isCyclic() !=1) and (list_of_visited_nodes2.count(key
    [0]) ==1) and (list_of_visited_nodes2.count(key[1])==1):
        list_of_visited_nodes2.append(key[0])
        list_of_visited_nodes2.append(key[1])
        final_cost2.append(value)
    else:
        g.delEdge(key[0]-1,key[1]-1)
connect = []
for i in list_of_visited_nodes2:
    if list_of_visited_nodes2.count(i) == 1:
        connect.append(i)
```

Program 4.2: Drugi dio koda.

U drugom dijelu algoritma prolazimo kroz rječnik sortiranih težina bridova od čvorova koji još nisu dodani u rutu. Potrebno je svaki od preostalih čvorova dodati u graf i provjeriti hoće li postojati ciklus. Osim toga, potrebno je osigurati da se dodani čvor ponovio točno jednom, kako ne bi postojala dva brida iz jednog čvora. Ako je to istina, dodani čvor će ostati u grafu, inače ćemo ga obrisati. Na kraju će nam u listi posjećenih čvorova ostati samo dva čvora, prvi i posljednji koji su se ponovili samo jednom i njih na kraju trebamo spojiti. To je prikazano u trećem dijelu koda.

```
if (connect[1],connect[0]) in sorted_weights and (connect[0],
    connect[1]) in sorted_weights:
    if sorted_weights[(connect[0],connect[1])]< sorted_weights[(
    connect[1],connect[0])]:
        list_of_visited_nodes2.append(connect[0])
        list_of_visited_nodes2.append(connect[1])
        final_cost2.append(sorted_weights[(connect[0],connect[1])
    ])
    else:
        list_of_visited_nodes2.append(connect[1])
        list_of_visited_nodes2.append(connect[0])
        final_cost2.append(sorted_weights[(connect[1],connect[0])
    ])
```

```
10 else:
11     if (connect[1],connect[0]) in sorted_weights:
12         list_of_visited_nodes2.append(connect[1])
13         list_of_visited_nodes2.append(connect[0])
14         final_cost2.append(sorted_weights[(connect[1],connect[0])
       ])
15     else:
16         list_of_visited_nodes2.append(connect[0])
17         list_of_visited_nodes2.append(connect[1])
18         final_cost2.append(sorted_weights[(connect[0],connect[1])
       ])
19
20 visited_nodes_and_final_cost.append(list_of_visited_nodes2)
21 visited_nodes_and_final_cost.append(sum(final_cost2))
22
23 return visited_nodes_and_final_cost
```

Program 4.3: Treći dio koda.

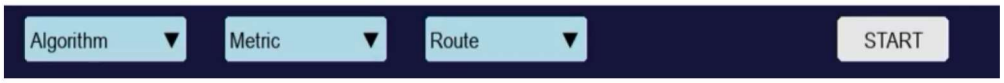
U zadnjem dijelu koda provjerava se postoje li u rječniku čvorovi A-B i B-A, te odabire koji ćemo dodati u listu posjećenih čvorova ovisno o tome koji od njih se nalazi u listi. Funkcija na kraju vraća listu koja sadrži posjećene čvorove i završni trošak. Završni trošak predstavlja sumu svih težina bridova koje je putnik prešao. Krajnji rezultat dan je na slici 4.3.

Slika 4.2: Prikaz nastalog grafa nakon prvog dijela pohlepnog algoritma.

Slika 4.3: Krajnji rezultat pohlepnog algoritma.

4.2 Grafičko sučelje

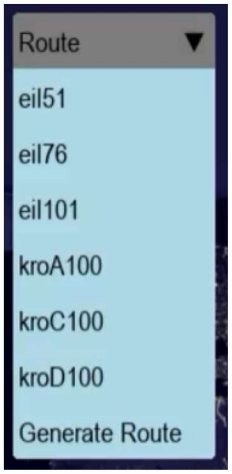
Za implementaciju grafičkog sučelja korišten je Pygame. Pygame je skup modula namijenjen izradi računalnih igrica. Podržava rad s grafikom, zvukom i input metodama. Pomoću Pygame-a implementirani su padajući izbornici prikazani na slici 3.3.



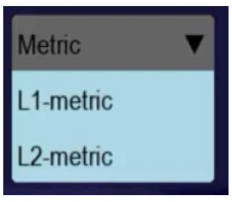
Slika 4.4: Izbornici u Pygame-u.

U prvom izborniku moguće je odabrati algoritam koji će se koristiti za rješavanje problema trgovačkog putnika. Drugi izbornik sadrži L1 i L2 metriku i treći izbornik sadrži instance problema. Prvih šest ruta zadane su unaprijed, dok se u zadnjoj generira 25 slučajno odabranih točaka ako ju odaberemo i pokrenemo neki od algoritama.

Slika 4.5: Izbornik za algoritme.



Slika 4.6: Izbornik za rute.



Slika 4.7: Izbornik za metrike.

Literatura

[1] D. Jukić, *Realna analiza*, Sveučilište Josipa Jurja Strossmayera u Osijeku - Odjel za matematiku, Osijek, 2020.

[2] G. Dantzig, R. Fulkerson, S. Johnson, *Solution of a Large-Scale Traveling-Salesman Problem*, Journal of the Operations Research Society of America, INFORMS,Vol. 2, No. 4 (Nov., 1954)), 393-410.

[3] `https://hr.education-wiki.com/6541278-what-is-a-greedy-algorithm`.

[4] `https://www.manhattanmetric.com/blog/2012/08/what-is-a-manhattan-metric.html`.

[5] `https://www.math.uwaterloo.ca/tsp/history/`.

[6] `http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/`.

[7] N. L. Biggs, E. K. LLoyd, and R. J. Wilson, *Graph Theory 1736-1936*,Clarendon Press, Oxford, 1976.

[8] P. Freguglia, M. Giaquinta, *The Early Period of the Calculus of Variations*, Springer International Publishing, Cham, 2016.