Rješavanje problema trgovačkog putnika upotrebom algoritma najbližeg susjeda

Javor, Ema

Sveučilište Josipa Jurja Strossmayera u Osijeku

Odjel za matematiku

Sveučilišni preddiplomski studij Matematika i računarstvo

Ema Javor

Rješavanje problema trgovačkog putnika upotrebom algoritma najbližeg susjeda

Završni rad

Mentor:

doc. dr. sc. Domagoj Ševerdija

Osijek, 2023

Sažetak

U ovom završnom radu obradit ću problem trgovačkog putnika. To je je problem diskretne i kombinatorne optimizacije u kojem je cilj pronaći najkraću rutu tako da se svi dani gradovi obiđu jednom i nakon toga vrati u početni grad. Naši gradovi su prikazani pomoću koordinata, a bridovi koji ih spajaju imaju unaprijed predodređene težine koje predstavljaju udaljenost između gradova. Svrha rješenja ovog problema jest pronaći najoptimalniji, odnosno da suma težina bridova koje smo prošli bude najmanja moguća. Za rješavanje problema trgovačkog putnika najčešće se koriste heuristički algoritmi kao što su pohlepni algoritam, algoritam najbližeg susjeda, algoritam nasumičnog umetanja, 2-opt, 3-opt... U ovom radu je implementiran i opisan algoritam najbližeg susjeda te je također prikazan izgled napravljenog grafičkog sučelja.

Ključne riječi

problem trgovačkog putnika, algoritam najbližeg susjeda

Solving the travelling salesman problem using the nearest neighbor algorithm

Abstract

In this final thesis, I will address the traveling salesman problem. This is a problem of discrete and combinatorial optimization in which the objective is to find the shortest route that visits all given cities once and then returns to the starting city. Our cities are represented using coordinates, and the edges connecting them have predetermined weights representing the distances between cities. The purpose of solving this problem is to find the most optimal solution, i.e., to minimize the sum of the weights of the edges traveled. Heuristic algorithms such as the greedy algorithm, the nearest neighbor algorithm, the random insertion algorithm, 2-opt, 3-opt, etc., are commonly used to solve the traveling salesman problem. In this paper, the nearest neighbor algorithm is implemented and described, and the appearance of the created graphical interface is also presented.

Key words

traveling salesman problem, nearest neighbor algorithm

Sadržaj

1 | Uvod

Problem trgovačkog putnika ili skraćeno TSP (engl. *Travelling Salesman Problem*) je problem diskretne i kombinatorne optimizacije. U klasičnom simetričnom TSP-u, gdje je udaljenost između dvaju gradova ista u oba smjera, složenost se može izraziti kao O(n!), gdje je "n" broj gradova. Ova složenost predstavlja faktorijelni rast i vrlo brzo raste s povećanjem broja gradova. Putniku su dane koordinate kao gradovi koje treba obići i udaljenosti između tih gradova. Traži se put kojim će putnik obići svaki od tih gradova samo jednom tako da zbroj udaljenosti bude najmanji moguć odnosno da rješenje bude najoptimalnije. U literaturi ovaj problem je moguće pronaći u brojnim radovima. Na prvi pogled ovaj se problem i ne čini preteškim, ali ako se uzme u obzir da ima faktorijelnu složenost, tj. da se najkraći put između N gradova nalazi negdje unutar prostora stanja koji je velik N!/2N shvatimo da je pronalaženje rješenja ipak kompleksnije nego što se čini. 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 pronalaženje rješenja koristit ćemo algoritam najbližeg susjeda. Osnovna ideja algoritma najbližeg susjeda je da počne od proizvoljnog početnog čvora (najčešće prvi grad) i zatim iterativno bira najbližeg susjednog čvora koji još nije posjetio. Algoritam ponavlja ovaj korak sve dok ne posjeti sve čvorove i na kraju se vraća na početni čvor.

Poglavlje 2 opisuje ideju problema trgovačkog putnika te detaljnije o njegovom nastanku. Također se osvrćemo na metrike koje su nužne pri pronalaženju udaljenosti za rješavanje porblema. Poglavlje 3 se bavi algoritmom najbližeg susjeda koji je jedan od algoritama koje možemo koristiti kako bi dobili rješenje ovog problema. Ulazimo detaljnije u njegov nastanak te ga opisujemo pseudokodom i primjerom iz projektnog dijela ovog rada. Zatim u 4. poglavlju prikazujemo grafičko sučelje i implementaciju problema trgovačkog putnika te opisujemo izgled intsanci problema koje smo koristili.

2 | Problem trgovačkog putnika

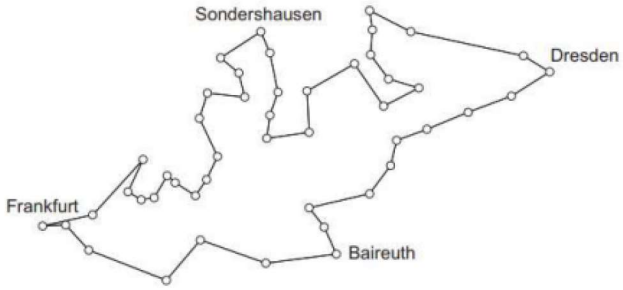
Problem trgovačkog putnika je poznati problem u teoriji optimizacije i računalnoj znanosti. Opisan je tako da imamo skup gradova koje nekad nazivamo čvorovima, a svaki par gradova povezan je cestom ili rutom s određenom udaljenosti koju također nazivamo i težinom. Trgovac želi posjetiti svaki grad iz tog skupa točno jednom i vratiti se u početni grad. Cilj je pronaći najkraći mogući put koji će omogućiti trgovcu da posjeti sve gradove i vrati se na početak. Pri rješavanju se većinom koriste grafovi svaki grad označava vrh grafa, početni grad je početna točka, dok su bridovi grafa putovi između dva vrha koji imaju preodređenu težinu. TSP je poznat po svojoj složenosti, jer broj mogućih rješenja eksponencijalno raste s brojem gradova. Ovaj problem ima mnoge praktične primjene izvan trgovine, kao što su rute dostave, planiranje putovanja, dizajn mikročipova, genetski algoritmi, itd. Svoj prvi matematički oblik i formalnu definiciju dobio je u prvoj polovini 20. stoljeća. Često se pripisuje Oskaru Zariskomu i Hassleru Whitneyju da su ga formalizirali u 1930-ima.

2.1 Povijest problema trgovačkog putnika

Nije poznato tko je definirao i kada je definiran problem trgovačkog putnika. Taj problem bio je poznat ljudima puno prije nego li je postao popularan kao matematički problem. Oblik matematičkog problema trgovačkog putnika proučavao je i švicarski matematičar Euler koji je na primjeru skakača na šahovskoj ploči razmatrao kako bi skakač mogao posjetiti sva 64 polja samo jednom. Prvo poznato spominjanje problema trgovačkog putnika je u priručniku njemačkog trgovačkog putnika Commis-Voyageur iz 1832. godine u kojemu je problem detaljno opisan riječima, ali ne i matematički. Priložio je 5 ruta kroz Njemačku i Švicarsku, no 4 rute su ipak sadržavale povratak u neki već posjećeni grad.

Matematičar Karl Menger počeo je proučavati opći oblik problema trgovačkog putnika te ga je 1930. godine i matematički opisao. Američki matematičar Hassler Whitney zaslužan je za uvođenje pojma „problem trgovačkog putnika“. Tijekom 1940-ih problem su proučavali brojni matematičari te 1950-ih i 1960-ih problem je postajao sve popularniji među europskim i američkim znanstvenicima. 1954. godine Dantzig, Fulkerson i Johnson objavili su opis metode za rješavanje problema i dali primjer od 49 instanci. Problem trgovačkog putnika privukao je mnogo pažnje i postao popularan u zajednici računalne znanosti. Također su bile organizirane i nagrade za pronalazak najboljih rješenja za određene instance TSP-a. Danas je TSP još uvijek aktivno područje istraživanja s mnogo primjena u stvarnim pro-

blemima logistike, rute dostave, planiranja putovanja i mnogim drugim područjima. Algoritmi za rješavanje problema trgovačkog putnika i dalje se razvijaju i poboljšavaju kako bi se pronašla optimalnija ili bolja aproksimacija za različite instance ovog problema.



Slika 2.1: Cammis- Voyageur, 1832.

2.2 Metrike

Da bismo izračunali udaljenost između gradova potrebno nam je izabrati koju metriku ćemo koristiti. Metrika u matematičkom smislu je funkcija ili sustav koji mjeri razdaljinu, sličnost ili odnos između elemenata u prostoru ili skupu podataka. Pri pokretanju našeg rada za izbor smo ponudili dvije metrike, Euklidsku i Manhattan.

2.2.1 Manhattan metrika

Manhattan metrika, također poznata kao L1 udaljenost, je metrika koja se koristi za mjerenje udaljenosti između dvije točke u prostoru. Naziv "Manhattan" dolazi od sličnosti sa načinom kretanja u ulicama Manhattana, gdje se putuje pravocrtno po vodoravnim i okomitim ulicama.

Manhattanska metrika mjeri udaljenost između dvije točke tako da zbraja apsolutne razlike njihovih koordinata. Formalno, udaljenost između dvije točke u 2-dimenzionalnom prostoru dana je sa:

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

Manhattanska metrika ima svoje prednosti i primjene u različitim područjima, ali treba razmotriti i druge metrike, poput Euklidske metrike, ovisno o konkretnim potrebama problema mjerenja udaljenosti ili sličnosti.

2.2.2 Euklidska metrika

Euklidska metrika, također poznata kao L2 udaljenost, je metrika koja se koristi za mjerenje udaljenosti između dvije točke u prostoru. Ova metrika temelji se na Pitagorinom teoremu iz geometrije. To znači da se udaljenost između dvije

točke izražava kao hipotenuza pravokutnog trokuta, a kvadrati razlika između koordinata čine kvadrate kateta tog trokuta. Udaljenost između dvije točke u 2-dimenzionalnom prostoru dana je sa:

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

Formulu za udaljenost prvi je objavio Alexis Clairaut 1731.g. Unatoč tome što ju je on objavio, ime je dobila po grčkom matematičaru Euklidu koji se prvi bavio problemom računanja udaljenosti.

3 | Algoritam najbližeg susjeda

3.1 Povijest i ideja algoritma najbližeg susjeda

Algoritam najbližeg susjeda nije povezan s jednim konkretnim pojedincem koji ga je osmislio, već je razvijan i primjenjivan tijekom vremena u kontekstu problema trgovačkog putnika i drugih sličnih problema optimizacije.

Različite verzije algoritma najbližeg susjeda i sličnih heurističkih pristupa koristile su se tijekom mnogo desetljeća u različitim kontekstima, a doprinosi su došli iz različitih izvora u područjima kao što su računalna znanost, operacijska istraživanja, matematika i inženjering.

Algoritam najbližeg susjeda (eng. Nearest Neighbor Algorithm) je heuristički algoritam koji se često koristi za rješavanje problema trgovačkog putnika i sličnih problema pronalaženja bliskih rješenja u kontekstu problema putovanja ili obilaska grafa.

Koncept ovog algoritma je zapravo vrlo jednostavan za shvatiti. Osnovna ideja algoritma najbližeg susjeda je da počne od proizvoljnog početnog čvora (najčešće prvi grad) i zatim iterativno bira najbližeg susjednog čvora koji još nije posjetio. Algoritam ponavlja ovaj korak sve dok ne posjeti sve čvorove i na kraju se vraća na početni čvor.

Ključna prednost ovog algoritma je njegova jednostavnost i brzina izvođenja. Međutim, važno je napomenuti da algoritam najbližeg susjeda ne jamči uvijek optimalno rješenje problema trgovačkog putnika. Može pronaći suboptimalna rješenja u nekim slučajevima jer odabire najbližeg susjeda bez razmatranja cijele rute unaprijed.

3.2 Pseudokod algoritma najbližeg susjeda

Algoritam 1 Algoritam najbližeg susjeda

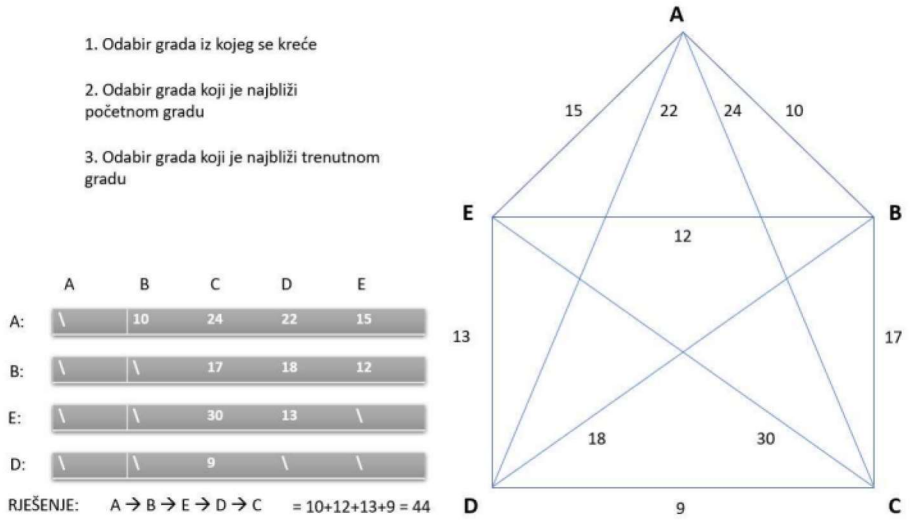
Ulaz: graf sa čvorovima C i težinama W
Izlaz: ruta

1: Inicijaliziraj praznu rutu R
2: Odaberi početni čvor kao trenutni čvor, na primjer, početni grad
3: Dodaj početni čvor u rutu R i označi ga kao posjećenog
4: **Sve dok** postoji neposjećeni gradovi u C: **čini**
5: Nađi neposjećeni grad s najmanjom težinom (najbližeg susjeda) iz trenutnog čvora
6: Dodaj taj grad u rutu R i označi ga kao posjećenog
7: Postavi trenutni čvor na taj grad
8: Vrati se na početni čvor kako biste zatvorili ciklus
9: R je rješenje problema trgovačkog putnika

U ovom pseudokodu, ključna ideja je iterativno odabirati najbližeg neposjećenog susjeda iz trenutnog čvora dok se svi gradovi ne posjete. Na kraju se ciklički vraćamo na početni čvor kako bismo zatvorili rutu.

3.3 Primjer algoritma najbližeg susjeda

Na slici 3.1 možemo vidjeti ilustraciju primjera primjene algoritma njabližeg susjeda. Prikazan nam je graf sa svim mogućim bridovima koji su ponuđeni i njihovim težinama. Cilj je naravno obići svih 5 čvorova tako da zbroj težina s bridova bude najmanji moguć.



Slika 3.1: Ilustrirani primjer algoritma najbližeg susjeda

Proizvoljno smo za početni brid izabrali čvor A i tražimo koji brid koji uključuje čvor A ima najmanju težinu. Kao što vidimo to je brid s težinom 10 koji završava u čvoru B. Iterativno ponavljamo taj postupak iz čvora B i sljedećih čvorova. Nakon što prođemo sve čvorove, zatvaramo rutu početnim čvorom i dobijemo rutu A - B - E - D - C - A kao rješenje.

3.4 Implementacija algoritma najbližeg susjeda

U projektnom dijelu rada implementirali smo algoritam najbližeg susjeda koji rješava problem trgovačkog putnika na način da za varijable funkcija *NearestNeighbour* koja je prikazana na Slici 3.2., prima informacije o međusobnim udaljenostima između svaka dva čvora, ukupnom broju čvorova i varijablu *node* koja označava proizvoljni početni čvor. Zatim određujemo da se funkcija ne završava dok prijeđeni put ne sadržava manje čvorova nego što ih ima u ukupnom broju čvorova koji je funkcija primila. Kao što smo već vidjeli u pseudokodu, funkcija pronalazi najkraću udaljenost do slijedećeg čvora iz onog čvora u kojem smo trenutno. Slijedeći čvor ne smije biti već posjećen. Informacije o udaljenostima dobivamo iz varijeble *weights*, a udaljenosti koje sadrži prijeđeni put spremamo u listu *costNearestNeighbour*. Kada prođemo kroz sve čvorove naš početni čvor postaje završni i zaustavljamo algoritam.

```
tour_nearest_neighbour = []
costNearestNeighbour = []


def NearestNeighbour(node, weights, graph_len):

    weights_from_certain_node = {}
    if len(tour_nearest_neighbour) >= graph_len:
        return

    for key, value in weights.items():
        if (key[0] == node and key[1] not in tour_nearest_neighbour) or (key[1] == node and key[0] not in tour_nearest_neighbour):
            weights_from_certain_node[(key[0], key[1])] = value

    tour_nearest_neighbour.append(node)

    if len(tour_nearest_neighbour) != graph_len:
        closest_nodes = min(weights_from_certain_node, key=weights_from_certain_node.get)
        cost = weights_from_certain_node[closest_nodes]
        costNearestNeighbour.append(cost)

        if closest_nodes[0] == node:
            NearestNeighbour(closest_nodes[1], weights, graph_len)
        NearestNeighbour(closest_nodes[0], weights, graph_len)
```

Slika 3.2: Implementacija algoritma najbližeg susjeda

4 | Implementacija i vizualizacija problema trgovačkog putnika

U projektnom dijelu ovog završnog rada implementirali smo 3 različita algoritjma za rješavanje problema trgovačkog putnika, Pohlepni algoritam, 2-opt algoritam i algoritam najbližeg susjeda. Osim 3 različita algoritma ponuđene su bile dvije već opisane metrike, Manhattan i Euklidska metrika. Sve to bilo je prikazano sučeljem rađenom u Pygame-u dok je za implementaciju korišten Python.

4.1 Zadavanje instanci problema

Instance problema zadane su iz biblioteke TSPLib u kojoj se nalaze primjeri problema trgovačkog putnika. Primjeri se sastoje od popisa vrhova i težina između njih kao što možemo vidjeti na slici 4.1. Prvi i drugi broj označavaju vrhove koje dodajemo u graf, a treći broj označava težinu na bridu između dva navedena vrha. Izdvojili smo 6 instanci za koje implementirani algoritmi rješavaju problem: eil51, eil76, eil101, kroA100, kroC100, kroD100. Svaka od tih istanci također sadrži i datoteku u kojoj se nalazi optimalno rješenje koje je unaprijed matematički dokazano za taj primjer. Uz sve to također je dodan izbor da program generira problem od 25 gradova s nasumično generiranim koordinatama. Cilj toga je bio da se na manjem pirmjeru s manje čvorova prikaže koncept algoritma.

```
NAME : eil51
COMMENT : 51-city problem (Christofides/Eilon)
TYPE : TSP
DIMENSION : 51
EDGE_WEIGHT_TYPE : EUC_2D
NODE_COORD_SECTION
1 37 52
2 49 49
3 52 64
4 20 26
5 40 30
6 21 47
7 17 63
```

Slika 4.1: Primjer nekoliko točaka iz instance eil51

Jednostavnim regularnim izrazom (engl. regex) iz instanci problema uzimamo

gradove i pripadne koordinate te ih spremamo u rječnik pomoću kojeg zatim računamo međusobne udaljenosti između svaka dva grada korištenjem jedne od dvije implementirane metrike.

4.2 Implementacija algoritma najbližeg susjeda

Implementacija algoritma pratila je pseudokod već ranije opisan u odjeljku 3.2. Metoda koja se pokretala pri odabiru algoritma najbližeg susjeda primala je čvor, težine i ukupni broj čvorova odnosno duljinu grafa. Metoda je nasumično izabrala početni čvor i nadodala ga u praznu listu koja bilježi rutu te u istu listu iterativno dodavala ostale čvorove redoslijedom koji naš algoritam nalaže. Metoda je završavala kada bi duljina te liste i duljina grafa bila jednaka što bi označavalo da je ruta završila u početnom čvoru i svi čvorovi su posjećeni točno jednom.

4.3 Grafičko sučelje

Za implementaciju grafičkog sučelja korišten je Pygame. Pygame je popularna Python biblioteka koja se koristi za razvoj igara i multimedijalnih aplikacija. Ova biblioteka pruža programerima alate i funkcionalnosti za stvaranje igara, simulacija, vizualizacija i drugih multimedijalnih aplikacija. U početnom izborniku odabiremo koji algoritam želimo koristiti, kojom metrikom želimo računati udaljenost i za koju rutu želimo riješiti problem. Euklidska metriaka označena je s L2, a Manhattan s L1 kao što možemo vidjeti na Slici 4.2.

Slika 4.2: Početni izbornik projektnog dijela rada

Pristikom na gumb START započinje rješavanje i vizualizacija odabranog problema i poziva se funkcija *start*() koja prvo traži informacije koje je korisnik oda-

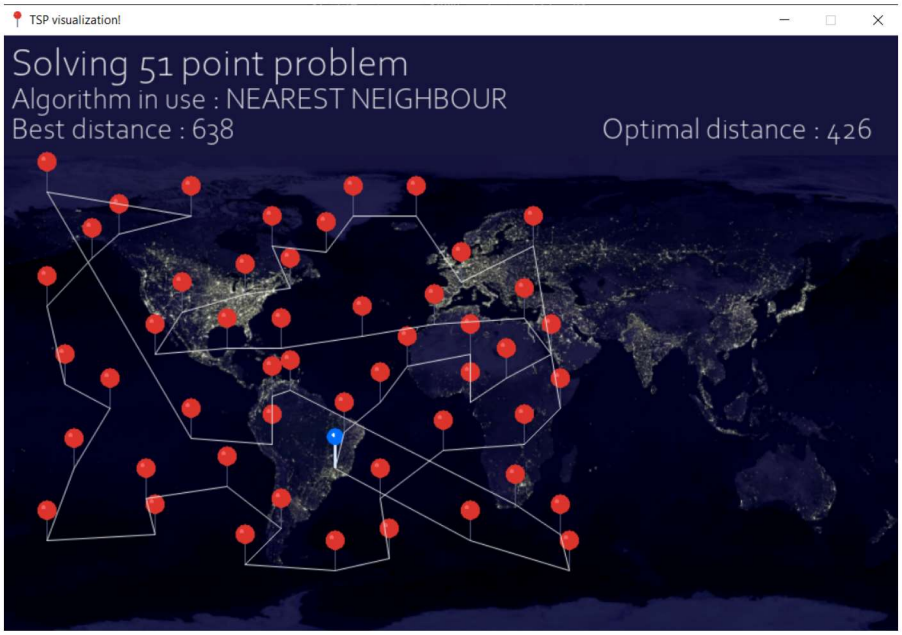
brao u padajućim izbornicima. Zatim dobijemo informacije o čvorovima kroz koje graf prolazi, njihovom redoslijedu i najoptimalnijoj udaljenosti. Te informacije spremaju se u listu pod imenom *infoForVisualization* i poziva se funkcija *drawCoords* kojom započinjemo crtanje čvorova i udaljenosti na sučelju. Funkcija prima varijable koje sadrže informacije o broju čvorova, međusobnoj udaljenosti čvorova i korisnikovom odabiru algoritma. Funkcijom *drawCoords*, čija je implementacija prikazana na Slici 4.3, crtaju se isključivo čvorovi, a zatim se poziva *drawGreedy*(), *drawNearestNeighbour*() ili *drawLocalSearch*() ovisno o izboru korisnika, kako bi se iscrtale linije koje prikazuju put koji smo izračunali da je najoptimalniji.

```
def drawCoords(final_coords, graph_len, ALGORITHM_IN_USE):

    WIN.blit(image_map, (0,120))

    solving_text = SOLVING_FONT.render(f"Solving {graph_len} point problem", 1, SOLVING_COLOR)
    WIN.blit(solving_text, (10,10))
    algorithm_text = DISTANCE_FONT.render(f"Algorithm in use : {ALGORITHM_IN_USE}", 1, SOLVING_COLOR)
    WIN.blit(algorithm_text, (10,50))

    max_xd = 0
    max_yd = 0
    for i in range(1, len(final_coords)+1):
        max_xd = max(final_coords[i][0], max_xd)
        max_yd = max(final_coords[i][1], max_yd)

    for i in range(1, len(final_coords)+1):
            xd = final_coords[i][0]
            yd = final_coords[i][1]

            multiplier_x = 9
            multiplier_y = 6

            while((max_xd*multiplier_x>880) or (max_yd*multiplier_y>600)):
                multiplier_x = multiplier_x/1.6
                multiplier_y = multiplier_y/1.5

            if (i==1):
                WIN.blit(STARTING_PIN, ((xd*multiplier_x)-20, (yd*multiplier_y)+80))
            else:
                WIN.blit(LOCATION_PIN, ((xd*multiplier_x)-20, (yd*multiplier_y)+80))
```

Slika 4.3: implementacija funkcije *drawCoords*

4.4 Primjer vizualizacije algoritma

Pri odabiranju izaberemo proizvoljnu rutu i metriku , dok za algoritam odaberemo algoritam 'Nearest neighbor'. Na primjeru sa Slike 4.4 koristili smo problem s 51 čvorom i dobili da je najoptimalnije rješenje korištenjem algoritma najbližeg susjeda težine 638 dok je matematičko najoptimalnije rješenje glasilo 426. Kod ovog algoritma nadodali smo plavu oznaku kako bi naglasili koji je početni čvor. Nakon što se algoritam provede i završi automatski je napravljena nova u tekstualna datoteka Route.txt u kojoj je ispisana odabrana ruta, udaljenost dobivena odabranim algoritmom, optimalna udaljenost, odabrana metrika, odabrani algoritam, broj gradova, koordinate gradova te rutu kojom treba posjećivati gradove kako bi postigli dobivenu udaljenost.



Slika 4.4: Prikazano rješenje rute eil51 korištenjem algoritma najbližeg susjeda

Literatura

[1] 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))

[2] A. Levitin, *Introduction to the Design and Analysis of Algorithms" by Anany Levitin*

[3] `https://en.wikipedia.org/wiki/Travelling_salesman_problem`

[4] A.R. Saiyed *The Traveling Salesman Problem, Indiana State University, 2012. dostupno na Internet stranici:* `http://cs.indstate.edu/~zeeshan/aman.pdf` .

[5] I. Dasović, *Rješavanje problema trgovačkog putnika uz pomoć genetskog algoritma, Elektrotehnika i informacijska tehnologija i Računarstvo, 2022.*

[6] `https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm`

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