SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN

Marko Kir

RAZVOJ APLIKACIJE ZA PLANIRANJE OPTIMALNE RUTE JAVNOG PRIJEVOZA

PROJEKT

UVOD U UMJETNU INTELIGENCIJU

Varaždin, 2024.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Marko Kir

Matični broj: 00161550

Studij: Informacijski i poslovni sustavi

RAZVOJ APLIKACIJE ZA PLANIRANJE OPTIMALNE RUTE JAVNOG PRIJEVOZA

PROJEKT

Mentor:

dr. sc. Bogdan Okreša Đurić

Varaždin, siječanj 2024.

Marko Kir

Izjava o izvornosti

Izjavljujem da je ovaj projekt izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor potvrdio prihvaćanjem odredbi u sustavu FOI Radovi

Sažetak

Rad se bavi razvojem aplikacije za planiranje optimalne rute javnog prijevoza, koristeći algoritme pretraživanja i automatskog planiranja. Teorijsko-metodološka polazišta uključuju pregled algoritama pretraživanja, njihove prednosti i ograničenja. Poseban naglasak je na primjeni ovih algoritama u kontekstu planiranja ruta javnog prijevoza. Glavne teze rada odnose se na implementaciju algoritama pretraživanja u razvoju aplikacije za planiranje ruta. Detaljno se analizira njihova učinkovitost i primjenjivost na stvarne scenarije javnog prijevoza. Zaključci rada naglašavaju važnost korištenja algoritama pretraživanja u planiranju ruta javnog prijevoza te prednosti implementacije takve aplikacije za krajnje korisnike. Kroz rad, uspješno je bilo pokazati praktičnu primjenu algoritama pretraživanja u kontekstu planiranja optimalnih ruta javnog prijevoza koristeći umjetnu inteligenciju.

Ključne riječi: optimalna; ruta; algoritam; pretraživanje; automatsko; planiranje; implementacija;

Sadržaj

1. Uvod

U ovom radu istražuje se razvoj aplikacije koja koristi algoritme pretraživanja i automatskog planiranja kako bi pružila optimalne rute javnog prijevoza. Fokus se stavlja na analizu primjene algoritama pretraživanja u planiranju ruta javnog prijevoza, uz istraživanje učinkovitosti i primjenjivosti u stvarnim scenarijima. Ovdje postavljamo temelje za detaljnu analizu implementacije algoritama pretraživanja u svrhu stvaranja efikasnih i optimalnih rješenja za korisnike javnog prijevoza. Tema ovog projekta odabrana je iz razloga što me zanimalo kako se uz pomoć algoritama pretraživanja i automatskog planiranja može doći do najefikasnije rute putovanja korištenjem javnog prijevoza.

2. Glavni dio teme

U ovom projektu fokusira se na primjenu algoritama pretraživanja, konkretno UCS (Uniform Cost Search) i A* (A-star) algoritama, u kontekstu automatskog planiranja optimalnih ruta javnog prijevoza. Prije nego što se detaljno opišu navedeni algoritmi, važno je definirati ključne pojmove i teoriju koja stoji iza formalizma umjetne inteligencije korištenog u projektu.

Umjetna inteligencija, u ovom kontekstu, predstavlja granu računalne znanosti koja se bavi razvojem sustava koji mogu simulirati inteligentno ponašanje. Formalizam umjetne inteligencije obuhvaća matematičke i logičke osnove koje omogućuju modeliranje problema i razvoj algoritama za njihovo rješavanje.

Algoritmi pretraživanja su ključni u automatskom planiranju ruta. UCS algoritam je oblik pretraživanja u širinu koji pronalazi put od početnog čvora do ciljnog čvora minimalne ukupne cijene. Ovaj algoritam pripada informiranim pretraživačima jer koristi informacije o cijenama prijelaza između čvorova.

A* algoritam kombinira prednosti neinformiranog pretraživanja (kao što je UCS) s dodatnim heurističkim informacijama o udaljenosti do cilja. Koristi se funkcija heuristike koja procjenjuje preostalu udaljenost do cilja kako bi efikasno usmjerio pretraživanje prema rješenju.

Teorija koja podržava ove algoritme uključuje koncepte grafova, gdje čvorovi predstavljaju stanja, a bridovi prijelaze između stanja. U ovom kontekstu, stanja predstavljaju različite lokacije ili čvorove u sustavu javnog prijevoza, a bridovi simboliziraju moguće prijelaze između tih lokacija.

Opisani formalizam temelji se na preciznim definicijama problema putovanja javnim prijevozom, čime se omogućuje implementacija i učinkovita primjena UCS i A* algoritama u razvoju aplikacije za planiranje optimalnih ruta.

3. Kritički osvrt

Osvrt na praktičnu izvedivost i primjenu algoritama pretraživanja, UCS i A* algoritama, u kontekstu razvoja aplikacije za planiranje optimalnih ruta javnog prijevoza ukazuje na nekoliko ključnih aspekata.

Prva bitna točka je praktična izvedivost implementacije algoritama. UCS algoritam, koji traži put minimalne ukupne cijene, i A* algoritam, kombinirajući informiranost s neinformiranim pretraživanjem, pružaju robustna rješenja. No, njihova učinkovitost može varirati ovisno o veličini i složenosti mreže javnog prijevoza. Implementacija može biti izazovna u velikim gradovima s kompleksnim sustavima prijevoza, gdje je brza pretraga po velikom prostoru stanja ključna za održavanje performansi.

Drugi ključni aspekt je primjenjivost rješenja u stvarnim scenarijima. Kako aplikacija ne uzima u obzir stvarne uvjete poput zagušenja prometa ili vremena čekanja na prijevozna sredstva, prednosti algoritama mogu biti ograničene. Također, korisničko iskustvo igra ključnu ulogu, pa je potrebno osigurati da su optimalne rute koje generira aplikacija praktične i korisne za krajnje korisnike.

Pitanje skalabilnosti također može utjecati na primjenu ovih algoritama u stvarnim uvjetima. Ako sustav ne može učinkovito rukovati velikim brojem korisnika istovremeno, to može dovesti do gubitka performansi i smanjenja korisničkog zadovoljstva.

U konačnici, u našem slučaju koristio se nerealističan skup podataka gdje nisu uključeni svi ključni faktori iz pravog svijeta da bi aplikacija mogla lakše raditi. U ovom primjeru korišteni su podaci o javnom prijevozu tramvajem u Zagrebu. Kako u pravoj bazi podataka ima puno unosa putovanja i termina, prava količina podataka je previše velika da bi se kod izvršio u nekom kratkom vremenu koje je potrebno korisniku. Zbog toga je baza podataka drastično smanjena na stanice i nasumično generirano vrijeme potrebno da se dođe od jedne stanice do druge kako bi se podaci mogli koristiti za naš primjer i prikaz rada algoritma pretraživanja i automatskog planiranja. Po tome možemo vidjeti da u realnosti za izradu ovakve aplikacije treba puno jače računalo ili više njih, te server/serveri koji bi upravljao/upravljali tim podacima, te bi tek onda ovaj projekt bio izvediv kako spada i bio primjenjiv u stvarnom svijetu u obliku aplikacije koju korisnici tramvaja grada Zagreba mogu koristiti. Na ovako maloj skali teško je izvesti takav projekt jer bi vrijeme potrebno da se obrade svi ti podaci bilo ogromno.

4. Opis implementacije

4.1. Biblioteke

```
import pandas as pd
import tkinter as tk
from tkinter import Label, Entry, Button, OptionMenu, StringVar, Listbox, messagebox
```

Isječak koda 1: Korištene biblioteke u projektu

4.1.1. Pandas

Pandas je biblioteka za programski jezik Python koja pruža efikasne i jednostavne strukture podataka za rad s podacima, posebno za analizu i manipulaciju tabličnih podataka. Uvoz pandas-a omogućuje korisnicima da koriste različite funkcionalnosti za rad s podacima, uključujući DataFrames, Series, i razne metode za filtriranje, grupiranje, i analizu podataka.

4.1.2. Tkinter

Tkinter je standardna biblioteka za izradu grafičkog korisničkog sučelja (GUI) u Pythonu. Pruža osnovne grafičke elemente i funkcionalnosti za izradu korisničkih sučelja. Uvoz tkinter-a omogućuje korisnicima kreiranje prozora, dugmadi, okvira i drugih GUI elemenata. Biblioteka olakšava izradu interaktivnih korisničkih sučelja za Python aplikacije.

4.2. Graf s podacima

4.2.1. Inicijalizacija grafa

```
graf = {
    "Zapadni kolodvor": [("Ulica Republike Austrije", 5), ("Ilica", 3), ("Trg bana
    ↪  Josipa Jelačića", 4)],
    "Trg bana Josipa Jelačića": [("Jurišićeva ulica", 2), ("Trg žrtava fašizma", 3),
    ↪  ("Glavni kolodvor Zagreb", 2)],
    "Trg žrtava fašizma": [("Zvonimirova ulica", 4), ("Borongaj", 5)],
}
```

Isječak koda 2: Kreiranje grafa s podacima

Kako je količina podataka iz pravih baza bila ogromna, kreiran je jednostavan graf po uzoru na neke tramvajske stanice grada Zagreba. Graf će kasnije biti korišten u algoritmima pretraživanja za automatsko planiranje optimalne rute javnog prijevoza.

4.2.2. Ažuriranje grafa - prvi dio

```
graf.update({
    'Zapadni kolodvor': [('Ulica Republike Austrije', 5),
                         ('Ilica', 3),
                         ('Trg bana Josipa Jelačića', 4)],
    'Trg bana Josipa Jelačića': [('Jurišićeva ulica', 2),
                                 ('Trg žrtava fašizma', 3),
                                 ('Glavni kolodvor', 2),
                                 ('Zapadni kolodvor', 4),
                                 ('Ilica', 4)],
    'Jurišićeva ulica':[],
    'Zvonimirova ulica':[],
    'Borongaj':[],
    'Trg žrtava fašizma': [('Zvonimirova ulica', 4),
                           ('Borongaj', 5),
                           ('Trg bana Josipa Jelačića', 3)],
    'Črnomerec': [('Ilica', 4),
                  ('Ulica Republike Austrije', 2)],
    'Ilica': [('Ulica Republike Austrije', 3),
              ('Jukićeva ulica', 2),
              ('Trg bana Josipa Jelačića', 4),
              ('Zapadni kolodvor', 3),
              ('Črnomerec', 4)],
    'Ulica Republike Austrije': [('Jukićeva ulica', 1),
                                 ('Vodnikova ulica', 4),
                                 ('Ilica', 3),
                                 ('Zapadni kolodvor', 5),
                                 ('Črnomerec', 2)],
    'Jukićeva ulica': [('Vodnikova ulica', 3),
                       ('Glavni kolodvor', 5),
                       ('Ilica', 2),
                       ('Ulica Republike Austrije', 1)],
    'Vodnikova ulica': [('Glavni kolodvor', 2),
                        ('Branimirova ulica', 3),
                        ('Jukićeva ulica', 3),
                        ('Ulica Republike Austrije', 4),
                        ('Savska cesta', 3)],
    'Glavni kolodvor': [('Branimirova ulica', 1),
                        ('Avenija Marina Držića', 4),
                        ('Draškovićeva ulica', 1),
                        ('Vodnikova ulica', 2),
                        ('Trg bana Josipa Jelačića', 2),
                        ('Jukićeva ulica', 5),
                        ('Savska cesta', 5)],
    'Branimirova ulica': [('Autobusni kolodvor', 2),
                          ('Glavni kolodvor', 1),
                          ('Vodnikova ulica', 3),
                          ('Avenija Marina Držića', 1)],
})
```

Isječak koda 3: Ažuriranje grafa - 1

4.2.3. Ažuriranje grafa - drugi dio

```
graf.update({
    'Avenija Marina Držića': [('Autobusni kolodvor', 3),
                              ('Avenija Vukovar', 4),
                              ('Kvaternikov trg', 3),
                              ('Most mladosti', 2),
                              ('Branimirova ulica', 1),
                              ('Glavni kolodvor', 4)],
    'Autobusni kolodvor': [('Avenija Vukovar', 1),
                           ('Žitnjak', 5),
                           ('Avenija Marina Držića', 3),
                           ('Branimirova ulica', 2)],
    'Avenija Vukovar': [('Žitnjak', 2),
                        ('Ulica grada Gospića', 3),
                        ('Gradsko poglavarstvo', 1),
                        ('Avenija Marina Držića', 4),
                        ('Autobusni kolodvor', 1),
                        ('Trg Dražena Petrovića', 5),
                        ('Savska cesta', 4)],
    'Žitnjak': [('Ulica grada Gospića', 4),
                ('Savišće', 5),
                ('Autobusni kolodvor', 5),
                ('Avenija Vukovar', 3)],
    'Ulica grada Gospića': [('Savišće', 2),
                            ('Žitnjak', 4),
                            ('Avenija Vukovar', 3)],
    'Savišće': [('Ulica grada Gospića', 2),
                ('Žitnjak', 5)],
    'Ljubljanica': [('Ozaljska ulica', 3)],
    'Ozaljska ulica': [('Tratinska ulica', 2),
                       ('Trg Dražena Petrovića', 4),
                       ('Ljubljanica', 3)],
    'Tratinska ulica': [('Trg Dražena Petrovića', 1),
                        ('Savska cesta', 3)],
    'Trg Dražena Petrovića': [('Savska cesta', 2),
                              ('Avenija Vukovar', 5),
                              ('Tratinska ulica', 1),
                              ('Ozaljska ulica', 4)],
    'Savska cesta': [('Avenija Vukovar', 4),
                     ('Vodnikova ulica', 3),
                     ('Glavni kolodvor', 5),
                     ('Tratinska ulica', 3),
                     ('Trg Dražena Petrovića', 2),
                     ('Savski most', 2),
                     ('Horvaćanska cesta', 3)],
    'Savski most': [('Savska cesta', 2)],
    'Draškovićeva ulica': [('Vlaška ulica', 4),
                           ('Glavni kolodvor', 1)],
})
```

Isječak koda 4: Ažuriranje grafa - 2

4.2.4. Ažuriranje grafa - treći dio

```
graf.update({
    'Vlaška ulica': [('Maksimirska cesta', 2),
                     ('Draškovićeva ulica', 4)],
    'Maksimirska cesta': [('Avenija Dubrava', 3),
                          ('Park Maksimir', 5),
                          ('Vlaška ulica', 2),
                          ('Kvaternikov trg', 4)],
    'Avenija Dubrava': [('Dubec', 4),
                        ('Maksimirska cesta', 3)],
    'Dubec': [('Avenija Dubrava', 4)],
    'Prečko': [('Horvaćanska cesta', 2)],
    'Horvaćanska cesta': [('Savska cesta', 3),
                          ('Prečko', 2)],
    'Gradsko poglavarstvo': [('Avenija Vukovar', 1)],
    'Kvaternikov trg': [('Maksimirska cesta', 4),
                        ('Avenija Marina Držića', 3)],
    'Park Maksimir': [('Maksimirska cesta', 5)],
    'Most mladosti': [('Avenija Dubrovnik', 3),
                      ('Avenija Marina Držića', 2)],
    'Avenija Dubrovnik': [('Muzej suvremene umjetnosti', 7),
                          ('Most mladosti', 3)],
    'Muzej suvremene umjetnosti': [('Sopot', 4),
                                   ('Avenija Dubrovnik', 7)],
    'Sopot': [('Muzej suvremene umjetnosti', 4)],
})
```

Isječak koda 5: Ažuriranje grafa - 3

U ovim djelovima koda nadodane su međusobno povezane tramvajske stanice. Kao što vidimo u kodu lijevo se nalazi naziv stanice kao naziv čvora, nakon ':' imamo popis stanica(čvorova) s kojima je povezan, te za svaku od te stanice imamo izmišljeno vrijeme potrebno da se dođe do nje, odnosno cijenu.

4.3. Uniform Cost Search - UCS algoritam

```
def ucs(graph, start, goal, heuristic=None):
visited = set()
queue = [(start, 0, [start])]

while queue:
    queue.sort(key=lambda x: x[1])
    node, cost, path = queue.pop(0)

    if node in visited:
        continue

    visited.add(node)

    if node == goal:
        return (cost, path)

    for child, child_cost in graph.get(node, []):
        if child not in visited:
            queue.append((child, cost + child_cost, path + [child]))

return None
```

Isječak koda 6: UCS kod

„Uniform Cost Search expand the node with low-cost path. It is implemented using the priority queue." [1, str. 2]

Prvo inicijlaiziramo skup i red sa visited: Skup za praćenje čvorova koji su već posjećeni. queue: Red koji sadrži trojke (čvor, trošak puta do tog čvora, lista puta do tog čvora). Zatim se pozicioniramo na početni čvor. Dodajemo ga u red queue s troškom 0 i putem koji sadrži samo početni čvor. Glavna petlja se izvršava dok red queue nije prazan. Na početku svake iteracije, red se sortira prema trošku puta, a zatim se uzima čvor s najmanjim troškom. Ako je čvor već posjećen, preskačemo ga i nastavljamo s idućim čvorom. Provjerava se ako je trenutni čvor jednak ciljnom čvoru, te ako je, vraćamo trošak i put do tog čvora.Ako nije, dodavamo susjeda u red tako da iteriramo kroz susjedne čvorove trenutnog čvora, provjeravajući jesu li već posjećeni. Ako susjed nije posjećen, dodajemo ga u red s ažuriranim troškom i putem. Za kraj petlje provjeravamo ako petlja završi, a ciljni čvor nije pronađen,te onda vraćamo None, što ukazuje na nedostatak puta do ciljnog čvora. Ovaj UCS algoritam dinamički prilagođava red prema troškovima, uvijek birajući čvor s najmanjim troškom. To osigurava da će pronaći najjeftiniji put do ciljnog čvora u grafu, što je karakteristično za uniformnu pretragu po cijeni.

4.3.1. Testiranje UCS

```
start_node = "Ilica"
goal_node = "Muzej suvremene umjetnosti"

result = ucs(graf, start_node, goal_node)
if result:
    print("UCS Cost:", result[0])
    print("UCS Path:", result[1])
else:
    print(f"Path from {start_node} to {goal_node} not found.")
```

Isječak koda 7: UCS test

Rezultat: UCS Cost: 20 UCS Path: [’Ilica’, ’Trg bana Josipa Jelačića’, ’Glavni kolodvor’, ’Branimirova ulica’, ’Avenija Marina Držića’, ’Most mladosti’, ’Avenija Dubrovnik’, ’Muzej suvremene umjetnosti’]

4.4. A* algoritam

```python
def astar(graph, start, goal, heuristic):
    node_info = {node: {'g': float('inf'), 'f': float('inf'), 'parent': None,
    ↪  'visited': False} for node in graph}

    node_info[start]['g'] = 0
    node_info[start]['f'] = heuristic[start]
    node_info[start]['visited'] = True

    S = {start}

    while S:
        u = min(S, key=lambda node: node_info[node]['f'])

        if u == goal:
            path = []
            while u is not None:
                path.insert(0, u)
                u = node_info[u]['parent']
            return (node_info[goal]['g'], path)

        S.remove(u)

        for w, t in graph.get(u, []):
            gt = node_info[u]['g'] + t
            if gt < node_info[w]['g']:
                node_info[w]['parent'] = u
                node_info[w]['g'] = gt
                node_info[w]['f'] = gt + heuristic[w]

                if not node_info[w]['visited']:
                    S.add(w)
                    node_info[w]['visited'] = True

    print("A* could not find a path.")
    return None

heuristic = {
    oznaka: sum(cvor[1] for cvor in djeca) for oznaka, djeca in graf.items()
}
```

Isječak koda 8: A* kod

Funkcija astar implementira A* algoritam za pronalaženje optimalnog puta u grafu. Ključni koraci algoritma uključuju inicijalizaciju informacija o čvorovima, postavljanje početnog čvora, korištenje prioritetskog reda za odabir čvorova, te iterativno proširivanje i ažuriranje čvorova prema procijenjenim troškovima. Ako ciljni čvor postane dostupan, algoritam rekonstruira put od cilja do početka i vraća trošak i put. Ako put nije pronađen, ispisuje se odgovarajuća poruka. A* algoritam koristi heurističku procjenu za učinkovito istraživanje grafa i pronalaženje

optimalnog rješenja. „A* employs an additive evaluation function f(n) = g(n) + h(n), where g(n) is the cost of the currently evaluated path from s to n and h is a heuristic estimate of the cost of the path remaining between n and some goal node.” [2, str. 3]

4.5. Kreiranje Grafičkog korisničkog sučelja i korištenje algoritama

4.5.1. Funkcije

```
def on_plan_route():
    algorithm = selected_algorithm.get()
    start_station = entry_start.get()
    goal_station = entry_goal.get()

    if start_station not in graf or goal_station not in graf:
        messagebox.showerror("Pogreška", "Nevaljanje stanice. Molimo Vas unesite valjane stanice.")
        return

    result = None

    if algorithm == "UCS":
        result = ucs(graf, start_station, goal_station)
    elif algorithm == "A*":
        result = astar(graf, start_station, goal_station, heuristic)

    if result:
        result_text.set(f"{algorithm} Cijena: {result[0]}\n{algorithm} Put: {result[1]}")
    else:
        result_text.set(f"Put od {start_station} do {goal_station} nije pronađen.")

def update_station_autocomplete(entry_widget, station_listbox):
    current_text = entry_widget.get()
    matching_stations = [station for station in graf if current_text.lower() in station.lower()]

    station_listbox.delete(0, tk.END)
    for station in matching_stations:
        station_listbox.insert(tk.END, station)

def select_autocomplete(entry_widget, station_listbox):
    selected_station = station_listbox.get(tk.ACTIVE)
    entry_widget.delete(0, tk.END)
    entry_widget.insert(0, selected_station)
```

Isječak koda 9: Funkcije

4.5.2. on_plan_route

Ova funkcija se poziva prilikom planiranja rute. Prvo dobiva odabrani algoritam, početnu i ciljnu stanicu. Provjerava jesu li unesene stanice valjane. Ako nisu, prikazuje se pogreška. Ovisno o odabranom algoritmu (UCS ili A*), poziva odgovarajuću funkciju (ucs ili astar) za izračun optimalnog puta između stanica. Ako je pronađen put, rezultati se postavljaju u varijablu result_text, inače se prikazuje poruka da put nije pronađen.

4.5.3. update_station_autocomplete

Ova funkcija ažurira popis stanica za automatsko dovršavanje teksta na temelju unesenog teksta u polje. Pretražuje stanice u grafičkom modelu (graf) koje odgovaraju unesenom tekstu. Prikazuje pronađene stanice u padajućem popisu (station_listbox). Funkcija select_autocomplete:

4.5.4. update_station_autocomplete

Ova funkcija se poziva kada se odabere stavka iz padajućeg popisa stanica. Dohvaća odabrano ime stanice i postavlja ga u odgovarajuće polje unosa (entry_widget), zamjenjujući prethodni tekst.

Ove funkcije čine korisničko sučelje za planiranje rute. Funkcija on_plan_route upravlja algoritmima za pronalaženje optimalnog puta, a funkcije update_station_autocomplete i select_autocomplete poboljšavaju korisničko iskustvo omogućavajući automatsko dovršavanje i odabir stanica.

4.5.5. Podešavanje Graphical User Interface-a

```
main_window = tk.Tk()
main_window.title("Planer optimalne rute javnog prijevoza tramvajem")

Label(main_window, text="Polazište:").grid(row=0, column=0, padx=10, pady=10)
Label(main_window, text="Odredište:").grid(row=1, column=0, padx=10, pady=10)
Label(main_window, text="Algoritam:").grid(row=2, column=0, padx=10, pady=10)

entry_start = Entry(main_window)
entry_start.grid(row=0, column=1, padx=10, pady=10)
entry_start.bind("<KeyRelease>", lambda event:
    ↪ update_station_autocomplete(entry_start, station_listbox_start))

entry_goal = Entry(main_window)
entry_goal.grid(row=1, column=1, padx=10, pady=10)
entry_goal.bind("<KeyRelease>", lambda event:
    ↪ update_station_autocomplete(entry_goal, station_listbox_goal))

algorithms = ["UCS", "A*"]
selected_algorithm = StringVar(main_window)
selected_algorithm.set(algorithms[0])
algorithm_menu = OptionMenu(main_window, selected_algorithm, *algorithms)
algorithm_menu.grid(row=2, column=1, padx=10, pady=10)

Button(main_window, text="Planiraj rutu", command=on_plan_route).grid(row=3,
    ↪ column=0, columnspan=2, pady=20)

result_text = tk.StringVar()
Label(main_window, textvariable=result_text).grid(row=4, column=0, columnspan=2,
    ↪ pady=10)

station_listbox_start = Listbox(main_window)
station_listbox_start.grid(row=0, column=2, padx=10, pady=10, rowspan=2)
station_listbox_start.bind("<ButtonRelease-1>", lambda event:
    ↪ select_autocomplete(entry_start, station_listbox_start))

station_listbox_goal = Listbox(main_window)
station_listbox_goal.grid(row=1, column=2, padx=10, pady=10, rowspan=2)
station_listbox_goal.bind("<ButtonRelease-1>", lambda event:
    ↪ select_autocomplete(entry_goal, station_listbox_goal))

main_window.mainloop()
```

Isječak koda 10: Kod za podešavanje GUI-a

Ovaj kod implementira jednostavno grafičko korisničko sučelje (GUI) u Pythonu pomoću Tkinter biblioteke. Sučelje je namijenjeno planiranju optimalne rute javnim prijevozom tramvajem.

Postoje dva polja za unos, jedno za polazišnu stanicu (entry_start) i drugo za odredišnu

```
print("Hello, World!")
```

Isječak koda 11: Primjer isječka koda

```
print("Ovo je preuzeti dio koda")
```

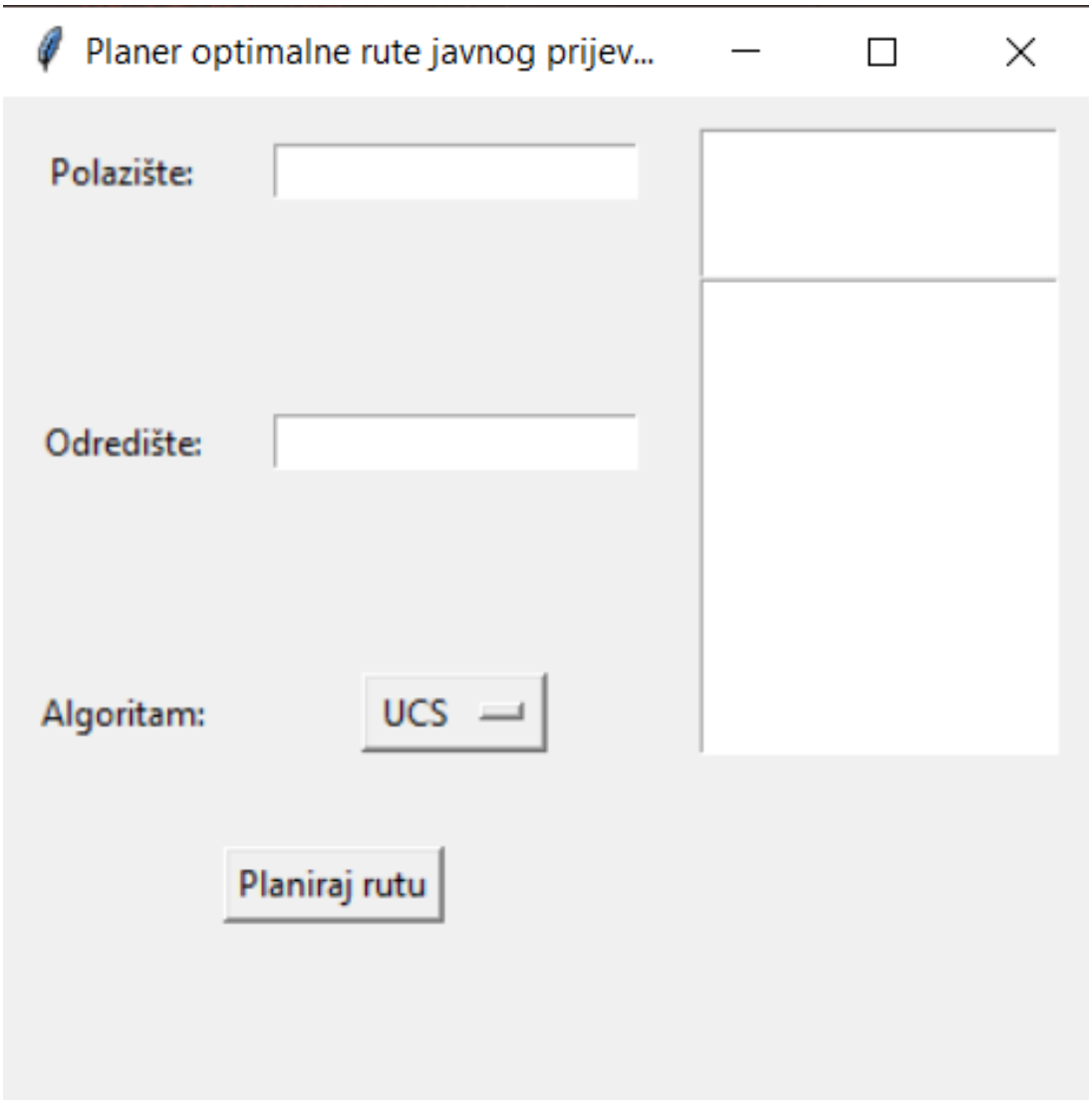
Isječak koda 12: Ovo je primjer koda koji je preuzet iz

stanicu (entry_goal). Korisnici unose stanice tramvajskog sustava u ova polja. Za oba polja unosa postoji padajući popis (station_listbox_start i station_listbox_goal), koji se prikazuje kada korisnici počnu unositi tekst. Funkcija update_station_autocomplete ažurira sadržaj padajućeg popisa ovisno o unesenom tekstu. Postoji padajući izbornik (algorithm_menu) s dva algoritma za odabir: UCS (Uniform Cost Search) i A*. Korisnici mogu odabrati algoritam koji će se koristiti za planiranje rute. Gumb "Planiraj rutu" pokreće funkciju on_plan_route kada je pritisnut. Ova funkcija koristi odabrani algoritam i unesene stanice kako bi pronašla optimalnu rutu. Zatim imamo tekstualni ispis (Label) koji prikazuje rezultate pretrage rute, uključujući trošak i put. Ovaj tekst se ažurira ovisno o rezultatima pretrage.

main_window.mainloop() pokreće glavnu petlju Tkintera, čime se omogućuje interakcija s korisničkim sučeljem. Ove komponente zajedno čine korisničko sučelje koje omogućuje planiranje optimalne rute tramvajem koristeći odabrane algoritme.

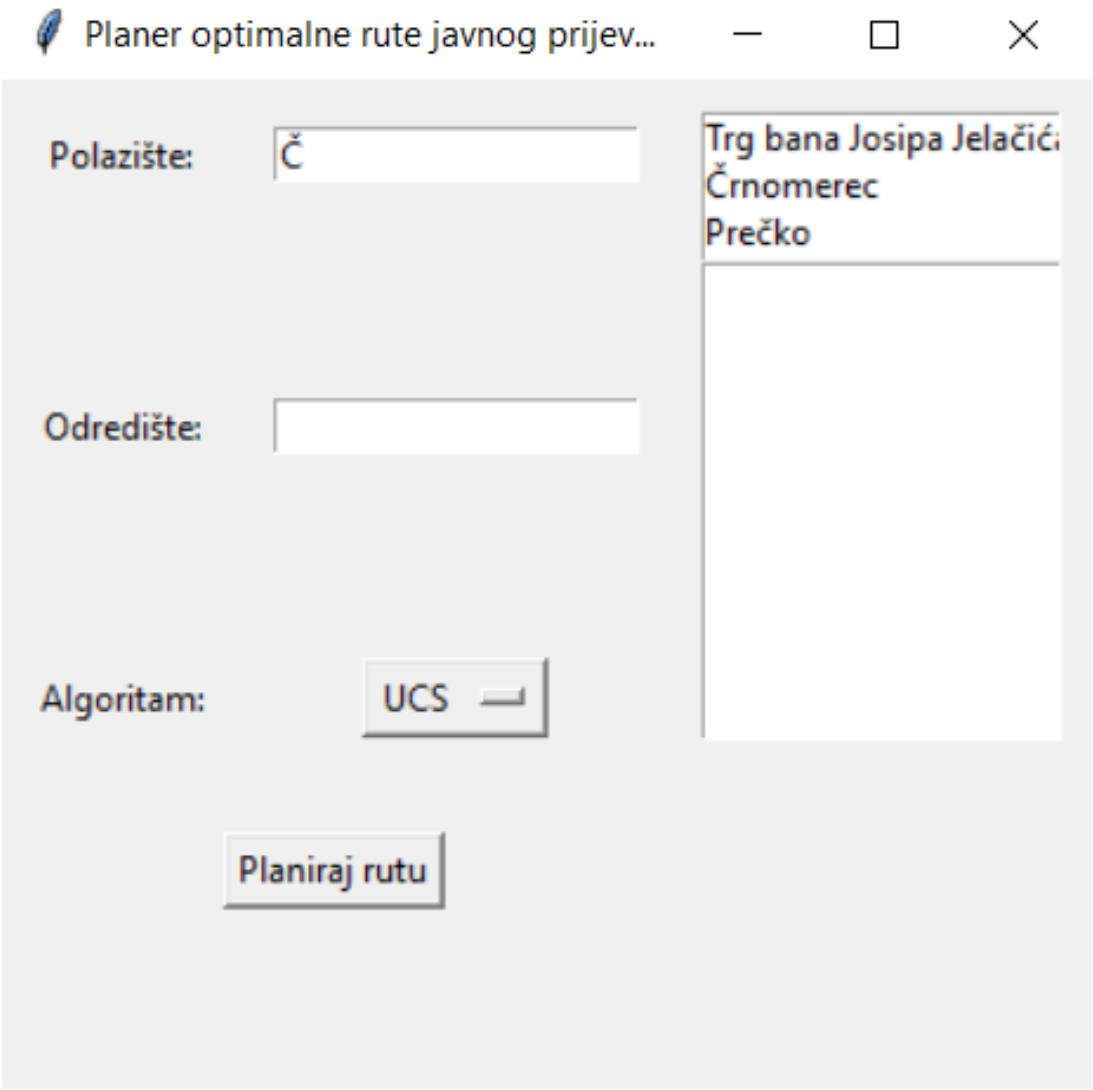
U slučaju preuzetog programskog koda, za isti je nužno potrebno naznačiti izvor, kao u isječku koda 12.

5. Prikaz rada aplikacije



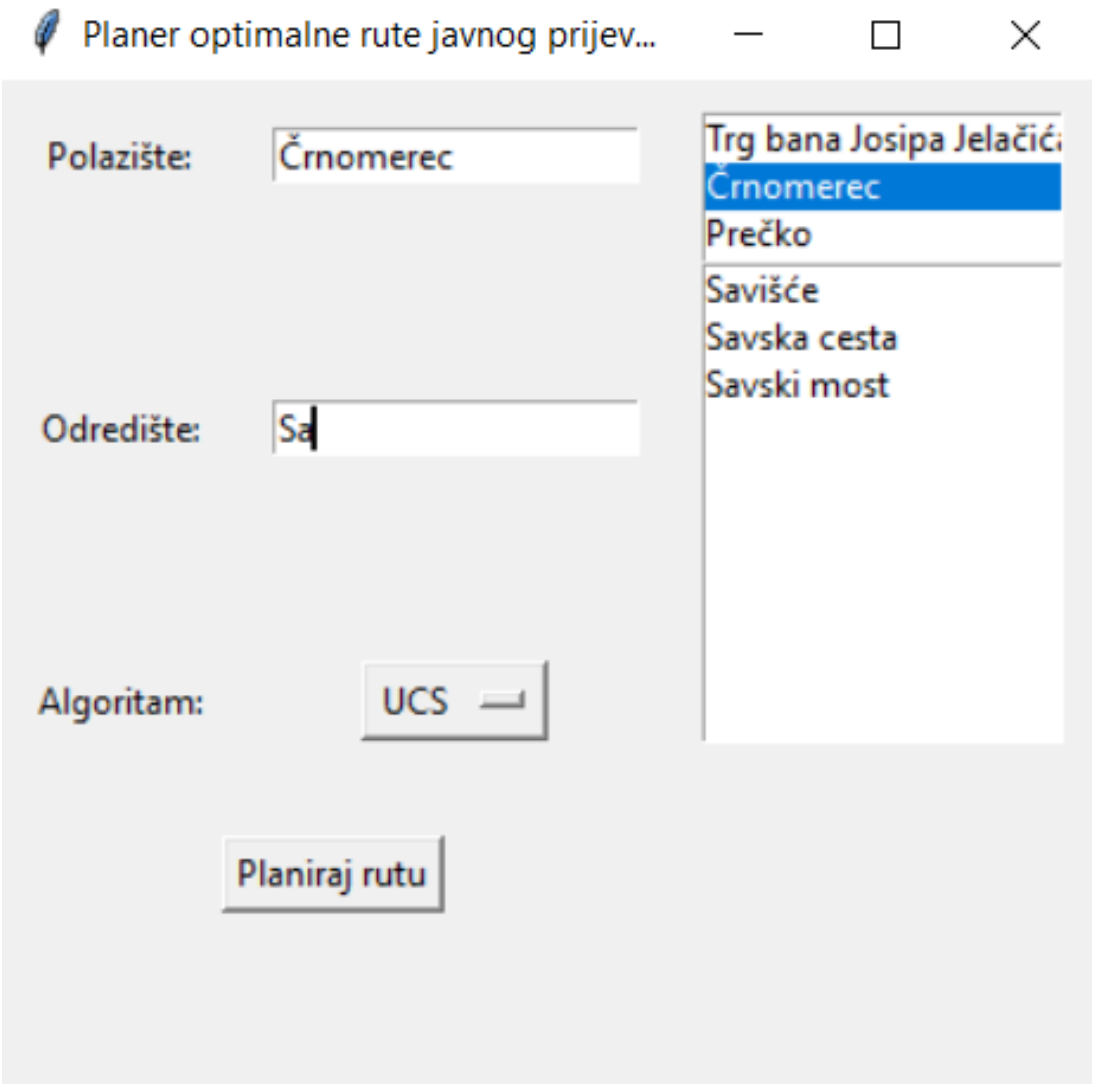
Slika 1: Sučelje aplikacije

Na ovoj slici vidimo sučelje aplikacije, odnosno kako sama aplikacija izgleda. Kao što vidimo imamo čelije za unos polazišta i odredišta, padajući izbornik za biranje algoritma, te prostore za automatsko popunjavanje teksta po onome što upišemo. Također imamo i gumb koji kad se pritisne ispisuje vrijeme potrebno za putovanje(cijenu), te najoptimalniju rutu što ćemo vidjeti u nastavku.

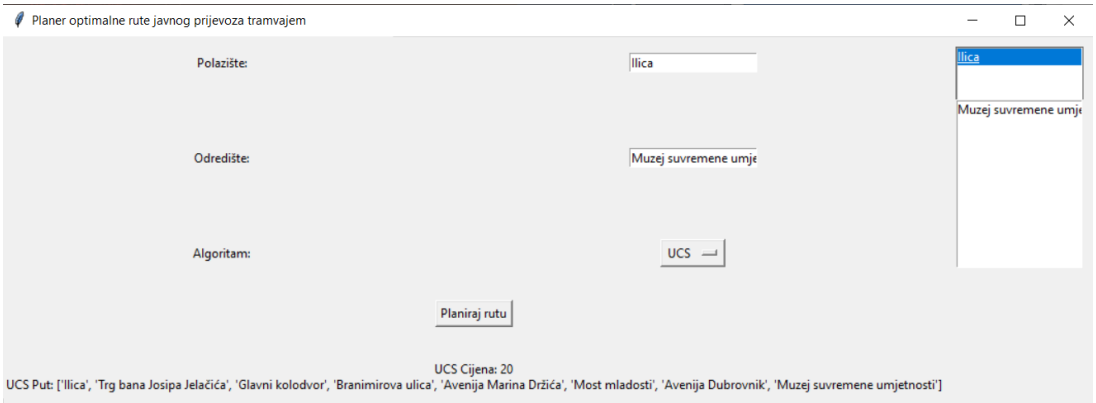


Slika 2: Automatsko dovršavanje polazišta

Nakon što se unese slovo aplikacija izbacuje prijedloge unosa, te ako se klikne dva puta na prijedlog on se automatski unese u čeliju za unos.

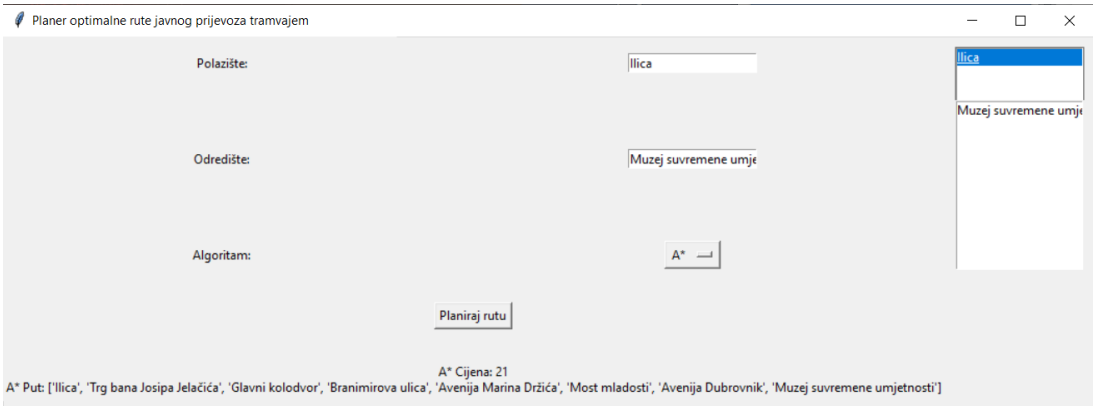


Slika 3: Automatsko dovršavanje odredišta



Slika 4: Ispis optimalne rute korištenjem UCS algoritma

Na ovoj slici vidimo prikaz najoptimalnije rute dobivene korištenjem UCS algoritma.



Slika 5: Ispis optimalne rute korištenjem A* algoritma

Na ovoj slici vidimo prikaz najoptimalnije rute dobivene korištenjem A* algoritma. U većini slučajeva će ruta biti jednaka za oba algoritma, no ima nekih kao ovaj na slici, gdje se malo razlikuje.

Podaci o svim bibliografskim jedinicama nalaze se u `lib.bib` datoteci u `BibLaTeX` formatu. Bibliografske jedinice korištene u ovom dokumentu su:

- članci iz časopisa [1], [2],

6. Zaključak

U knačnici, ovaj projekt je razvijen s ograničenjima koja proizlaze iz upotrebe nerealističnog skupa podataka, gdje nisu obuhvaćeni svi ključni faktori stvarnog svijeta kako bi aplikacija mogla efikasno funkcionirati. U konkretnom primjeru, korišteni su podaci o javnom prijevozu tramvajem u Zagrebu. U stvarnom svijetu, stvarna baza podataka sadržava obilan broj unosa putovanja i vremenskih termina, čija bi obrada zahtijevala znatno više resursa od onih dostupnih u okviru ovog projekta. Stoga je baza podataka značajno reducirana na stanice i nasumično generirano vrijeme potrebno za putovanje između njih, kako bi se podaci mogli prilagoditi primjeru i demonstrirati funkcionalnost algoritama pretraživanja i automatskog planiranja.

Ova redukcija podataka ukazuje na to da za stvarnu primjenu ove aplikacije, koju bi korisnici tramvaja grada Zagreba mogli koristiti, potrebno je ozbiljnije računalno i serversko okruženje. Izazov je u tome što bi u stvarnom svijetu bilo potrebno rukovati velikim obujmu podataka, za što su nužni snažniji računalni resursi i serverska infrastruktura. Tek uz takve uvjete, projekt bi mogao biti izvediv i primjenjiv u pravom svijetu, pružajući korisnicima korisničko iskustvo koje odražava stvarne uvjete i potrebe korisnika javnog prijevoza.

Popis literature

[1] M. J. Pathak, R. L. Patel i S. P. Rami, „Comparative analysis of search algorithms," *International Journal of Computer Applications*, sv. 179, br. 50, str. 40–43, 2018.

[2] R. Dechter i J. Pearl, „Generalized best-first search strategies and the optimality of A," *Journal of the ACM (JACM)*, sv. 32, br. 3, str. 505–536, 1985.

[3] K. V. „"Dnevne tramvajske linije u Zagrebu | Popis linija i trasa kojima prometuju", ZGportal Zagreb." [Online; accessed Jan. 14, 2024.] (Jan. 02, 2024.), adresa: https://www.zgportal.com/zginfo/dnevne-tramvajske-linije-u-zagrebu-popis-linija-i-trasa-kojima-prometuju/.

[4] OpenAI. „ChatGPT." [Online; accessed Jan. 14, 2024.] (2023.), adresa: https://chat.openai.com/.

Popis slika

Popis isječaka koda