

Akademija: Optimalne logistične rešitve

Akademija pri predmetu Matematično modeliranje

2023/24

V sodelovanju z:

OMNIOPTI d. o. o.

Avtorja:

Jan Kocet

Žan Luka Tomanič

Mentorji:

dr. Drago Bokal

Martin Pečar

Alen Vegi Kalamar

Vsebina

[1. Zahvala 2](#_Toc169185399)

[2. Uvod 2](#_Toc169185400)

[Akademija: Optimalne logistične rešitve 3](#_Toc169185401)

[1. Namestitev potrebščin 3](#_Toc169185402)

[1.1. SUMO – Simulation of Urban MObility by Eclipse 3](#_Toc169185403)

[1.2. Python 3](#_Toc169185404)

[1.3. PyCharm 4](#_Toc169185405)

[2. Priprava podatkov 4](#_Toc169185406)

[3. Algoritem 5](#_Toc169185407)

[3.1. Pretvarjanje podatkov 5](#_Toc169185408)

[3.2. Algoritem sestopanja 5](#_Toc169185409)

[3.3. Primerjava rešitev – izbira boljše 6](#_Toc169185410)

[3.4. Izvedba algoritma 6](#_Toc169185411)

[4. Prikaz rešitve 6](#_Toc169185412)

[5. Priloge 7](#_Toc169185413)

[5.1. Kazalo slik 7](#_Toc169185414)

## Zahvala

Zahvaljujeva se podjetju OmniOpti d. o. o. za ponujeno priložnost in g. Martinu Pečarju za predstavitev problema in vsega kar sva potrebovala za delo. Hvala tudi profesorju dr. Dragu Bokalu in asistentu Alenu Vegiju Kalamarju za nasvete in vodila.

## Uvod

Predstavljajmo si, da se ob 15:00 odpravljamo iz službe. Ker želimo čim prej priti domov, izberemo najhitrejšo oz. najkrajšo pot. Seveda pa s tem nismo edini. Naša vožnja domov se lahko zaradi prevelikih obremenitev nekaterih cest spremeni v čakanje v prometni konici. Seveda se temu ne moremo izogniti, če lahko domov pridemo le po eni poti. Kaj pa če imamo na voljo več poti, ki so si cenovno (čas, razdalja, gorivo …) zelo podobne? Z namenom da ne obtičimo v gneči, izberemo tisto pot, ki jo bo izbralo najmanj drugih uporabnikov cestnega omrežja.

V algoritmu pa ne bomo vzeli v obzir samo enega udeleženca na cestah, pač pa nas bo zanimalo celotno cesto omrežje na neki lokaciji in okolici. Ustvarili bomo torej algoritem, ki bo iz podatkov o cestnem omrežju vsem udeležencem na cesti, ki imajo za svoje »potovanje« na voljo več različnih (cenovno enakih oz. zelo podobnih) poti določil tisto, ki bo za naše celotno omrežje najmanj obremenjujoča.

Algoritem bomo zapisali v programskem jeziku PYTHON v okolju PyCharm. Uporabili bomo tudi odprtokodni paket SUMO – Simulator of Urban MObility, sistemsko orodje, namenjeno za delo z prometnimi omrežji. V sklopu akademije so tudi navodila za namestitev le-teh. Podatki, s katerimi bomo delali, bod zapisani tako, kot jih uporabljamo za SUMO, da prikažemo omrežje in simulacijo vseh poti.

Da bi se izognili (pre)veliki količini podatkov, bomo za boljšo predstavo in preglednost za algoritem uporabili omrežje, ki sva ga ustvarila midva. Prav tako bodo podani vsi udeleženci omrežja (avti) in vse njihove možne poti, za katere predpostavimo, da so vse enako ugodne. Tudi vse to bo obravnavano v akademiji.

Dotaknili pa se bomo tudi možnosti, da algoritem izvedemo na resničnem omrežju. Tega bi dobili na spletnem mestu OSM OpenStreetMap, na katerem najdemo kartografske podatke zastonj, za vsakogar.

# Akademija: Optimalne logistične rešitve

## Namestitev potrebščin

### SUMO – Simulation of Urban MObility by Eclipse

#### Prenos in namestitev SUMO:

1. Obiščite uradno spletno stran [SUMO](https://eclipse.dev/sumo/).
2. Pojdite na sekcijo "Download" in poiščite najnovejšo različico SUMO za Windows.
3. Prenesite "sumo-win64-\*.zip" datoteko (ali ustrezno za vaš operacijski sistem).
4. Razpakirajte preneseno stisnjeno datoteko na želeno lokacijo na vašem računalniku.

#### Nastavitev sistemske poti:

1. Odprite "Settings" in v iskalno vrstico vnesite "Environment Variables".
2. Kliknite "Edit the system environment variables".
3. V oknu "System Properties" kliknite na gumb "Environment Variables".
4. V sekciji "System variables" poiščite spremenljivko "Path" in jo uredite.
5. Kliknite "New" in dodajte pot do mape, kjer ste razpakirali SUMO (na primer C:\path\to\sumo\bin).
6. Kliknite "OK" za shranjevanje sprememb.

#### Preverjanje namestitve:

1. Odprite ukazno vrstico (Command Prompt) in vnesite sumo.
2. Če je namestitev uspešna, boste videli izpis pomoči za SUMO.

### Python

#### Prenos in namestitev Pythona:

1. Obiščite [uradno spletno stran Python](https://www.python.org/).
2. Pojdite na sekcijo "Downloads" in izberite "Download for Windows".
3. Prenesite najnovejšo različico Python-a.
4. Zaženite preneseno namestitveno datoteko.
5. V namestitvenem oknu izberite opcijo "Add Python to PATH" in nato kliknite "Install Now".
6. Počakajte, da se namestitev dokonča, in kliknite "Close".

#### Preverjanje namestitve:

1. Odprite ukazno vrstico (Command Prompt).
2. Vnesite python --version in pritisnite Enter.
3. Če je namestitev uspešna, boste videli številko različice Pythona.

#### Namestitev pip:

Pip je privzeto nameščen z novejšimi različicami Pythona. Če morate preveriti, ali je pip nameščen, sledite tem korakom:

1. Odprite ukazno vrstico (Command Prompt).
2. Vnesite pip --version in pritisnite Enter.
3. Če je pip nameščen, boste videli številko različice pip.

#### Namestitev dodatnih paketov:

Za namestitev dodatnih paketov uporabite pip:

1. Odprite ukazno vrstico (Command Prompt).
2. Vnesite pip install package\_name (kjer package\_name zamenjate z imenom želenega paketa) in pritisnite Enter.

### PyCharm

#### Prenos in namestitev PyCharm:

1. Obiščite [uradno spletno stran PyCharm](https://www.jetbrains.com/pycharm/download/).
2. Izberite različico, ki jo želite namestiti. Na voljo sta Community (brezplačna) in Professional (plačljiva) različica. Za večino uporabnikov je Community različica zadostna.
3. Prenesite namestitveno datoteko za Windows.
4. Zaženite preneseno namestitveno datoteko.
5. Sledite navodilom namestitvenega čarovnika. Priporočam, da obkljukate možnost "Add launchers dir to the PATH" in "Create Associations .py".
6. Kliknite "Install" in počakajte, da se namestitev dokonča.
7. Kliknite "Finish" za dokončanje namestitve.

#### Prvi zagon PyCharm:

1. Zaženite PyCharm.
2. Če vas vpraša za uvoz nastavitev, izberite "Do not import settings".
3. Sprejmite licenčne pogoje in sledite začetnim nastavitvam.

## Priprava podatkov

Kot že omenjeno v uvodu, so podatki o omrežju, ki ga bomo obravnavali ustvarjeni z namenom boljše preglednosti in lažje predstave algoritma.

* 1. Ustvarite mapo, v kateri boste delali s PyCharm.
  2. Poučite se o ustvarjanju omrežja za SUMO »na roke«.

<https://sumo.dlr.de/docs/Networks/PlainXML.html>

* 1. V svojo mapo prekopirajte datoteke: [.edg.xml](https://github.com/Kocetj/Sim_list/blob/main/sim_list.edg.xml), [.nod.xml](https://github.com/Kocetj/Sim_list/blob/main/sim_list.nod.xml), [.rou.xml](https://github.com/Kocetj/Sim_list/blob/main/sim_list.rou.xml), ki jih najdete na povezavah.
  2. Preglejte datoteke in narišite omrežje (ročno na list ali s knjižnico MatPlotLib, če jo poznate) za boljšo predstavo.
  3. Poučite se o poteh avtomobilov v našem omrežju (najdete jih v .rou.xml filih). Avti so označeni z id-ji, pri katerih celi del predstavlja »ime« avtomobila, decimalni pa označuje možno pot. Pot je zapisana kot zaporedje id-jev cest.

## Algoritem

### Pretvarjanje podatkov

1. V vaši mapi ustvarite main.py datoteko.
2. Poučite se o eXtensible Markup Language ([XML](https://www.geeksforgeeks.org/xml-basics/)).
3. Poučite se o slovarjih v Python ([Dictionary](https://www.geeksforgeeks.org/python-dictionary/)).
4. Poučite se o knjižnici [Minidom](https://www.geeksforgeeks.org/parse-xml-using-minidom-in-python/). Vključite jo v main.py datoteko.
5. Vključite tudi knjižnico *Math,* za lažje delo z decimalnimi števili.
6. Ustvarite funkcijo, ki bo za parameter dobila povezavo do *.rou.xml* datoteke. V njej z uporabo *.parse()* iz knjižnice Minidom uvozite podatke o avtomobilih in njihovih poteh in jih shranite kot slovar. Za lažje delo in preglednost naj bodo oblike*: {id-avtomobila: [ 1. možna pot, 2. možna pot, … ] }*. Možne poti so naj zapisane kot zaporedje id-jev cest (polje).

*Primer: Slika .rou.xml datoteke in izgled slovarja, ki ji pripada.*

*Slika, ki vsebuje besede besedilo, posnetek zaslona, pisava

Opis je samodejno ustvarjen*

Slika 1

**» {1: [[1, 2, 9], [1, 3, 6, 9], [1, 3, 8, 11]], 2: [[4, 7, 10, 11], [4, 5, 6, 9], [4, 5, 8, 11]]}«**

### Algoritem sestopanja

1. Poučite se o algoritmu sestopanja ([backtracking](https://www.geeksforgeeks.org/backtracking-algorithms/?ref=lbp)).
2. V main.py dodajte funkcijo *sestopanje*(), ki se bo »sprehodila« po možnih rešitvah našega problema.

Namig: funkcija naj izgleda kot: *sestopanje(moznePotiAvtov, avto, tmp, resitev).* moznePotiAvtov naj predstavlja slovar kot smo ga ustvarili s funkcijo pri 4.1. 6. S parametrom avto bomo označili avto, za katerega bomo vključili pot v tmp. Tmp nam bo predstavljal trenutno možno rešitev, ki jo bomo primerjali s spremenljivko rešitev.

Rešitev v psevdokodi:

ZA vsak *avto* v slovarju *moznePotiAvtov*:

ČE *avto* večji od dolžine *moznePotiAvtov*:

ČE *resitev* ni prazna:

posodobi *resitev* s trenutno vrednostjo *tmp*

ALI ČE *primerjajResitvi*(*tmp*, *resitev*):

posodobi *resitev* s trenutno vrednostjo *tmp*

KONEC

RETURN

KONEC

ZA vsako *pot* v *moznePotiAvtov*[*avto*]:

*tmp*[*avto*] = *pot*

Rekurzivno kliči funkcijo *sestopanje*() z novimi

parametri

*tmp*[*avto*] = None

KONEC

1. Za boljšo preglednost zapišite še funkcijo, s katero boste začeli sestopanje. Prejme naj slovar z možnimi potmi vseh avtov, vrne pa naj rešitev. V telesu določite spremenljivko rešitev kot prazen slovar, nato kličite funkcijo sestopanje. Podajte ji ustrezne vhodne podatke.

### Primerjava rešitev – izbira boljše

Ko funkcija sestopanje »pride do konca« tj. napolni *tmp* z neko potjo za vsak avto, potem ta slovar *tmp* primerjamo s slovarjem *resitev* in izberemo tistega, ki je za nas boljši. V našem primeru bomo boljšo rešitev izbrali tisto, ki bo imela najmanjši standardni odklon, če opazujemo s koliko avti je posamezna cesta zasedena. To seveda ni edina rešitev, je pa ena izmed možnih. Obstajajo tudi bolj optimalne rešitve tega problema, a so prezahtevne, da bi jih obravnavali v tej akademiji. Seveda pa ima bralec te akademiji proste roke za izbiro načina, s katerim bo določil, katera rešitev je boljša.

1. V main.py dodajte funkcijo primerjaj rešitvi.
2. Poučite se o merjenju in uporabi standardnega odklona.
3. V datoteko vključite knjižnico Numpy.
4. Ustvarite funkcijo *zasedenostCest*(), ki bo iz slovarja *tmp*, ki bo za vsak avto vseboval eno pot, določeno s poljem id-jev cest, vrnila slovar *zasednost*. V tem slovarju so naj ključi id-ji cest, vrednosti za vsak id-ceste, pa naj je število avtomobilov, ki bi se po tej cesti peljali.
5. Ustvarite funkcijo *primerjajResitvi(),* ki naj za parametra dobi slovarja tmp in rešitev. V njej za vsak parameter ustvarite slovar zasedenosti cest (funkcija iz prejšnje naloge). Primerjajte obe zasedenosti cest in s pomočjo knjižnice Numpy z metodama list() in std() ustrezno izberite in vrnite boljšo rešitev.

### Izvedba algoritma

1. V datoteko main.py dopolnite z vsem potrebnim, da izvedete algoritem.

## Prikaz rešitve

Na začetku smo rekli, da so podatki za avte in njihove poti zapisane v XML datotekah, da jih lahko uporabimo za prikaz s paketom SUMO. V tem poglavju se bomo dotaknili tudi tega. Prikazali bomo rešitev, do katere smo prišli.

1. Da bi lahko ustvarili prikaz oz. simulacijo s SUMO, potrebujemo v datoteki .rou.xml le tiste avte in njihove potmi, za katere smo se odločili v rešitvi. Ustvarite funkcijo, ki bo s pomočjo knjižnice Minidom in njenih metod datoteko .rou.xml spremenila tako, da bodo v njej ostali le podatki za avte in poti, ki smo jih izbrali v rešitvi.
2. Za simulacijo s SUMO potrebujemo .net.xml datoteko. Paket, ki smo ga prenesli ima vgrajeno metodo –netconvert, s katero iz datotek .edg.xml in .nod.xml ustvari .net.xml. S pomočjo [povezave](https://sumo.dlr.de/docs/Networks/PlainXML.html), tudi vi ustvarite potrebno .net.xml datoteko.
3. Za simulacijo potrebujemo tudi sumo.cfg datoteko. Podan bo primer izgleda te datoteke, vi ustvarite podobno za vas.

Slika, ki vsebuje besede besedilo, posnetek zaslona, pisava

Opis je samodejno ustvarjen

Slika 2

1. Vse je pripravljeno za prikaz simulacije. Odprite datoteko .sumo.cfg z SUMO-gui ali ustvarite novo Python [skripto](https://sumo.dlr.de/docs/TraCI/Interfacing_TraCI_from_Python.html#:~:text=First%20you%20compose%20the%20command%20line%20to%20start,sumoBinary%20%3D%20%22%2Fpath%2Fto%2Fsumo-gui%22%20sumoCmd%20%3D%20%5BsumoBinary%2C%20%22-c%22%2C%20%22yourConfiguration.sumocfg%22%5D), da zaženete simulacijo.

## Priloge

### Kazalo slik

[Slika 1 5](#_Toc169096832)

[Slika 2 7](#_Toc169096833)