Univerza v Ljubljani
Fakulteta za matematiko in fiziko

Poročilo projekta pri predmetu Finančni praktikum

Problem potujočega trgovca s premikajočimi se mesti vzdolž ene črte

Tinkara Čadež in Nika Furlan

Ljubljana, januar 2022

Kazalo

1 Uvod

V poročilu bova predstavili projektno nalogo, ki sva jo izdelali v okviru predmeta Finančni praktikum. Gre za problem potujočega trgovca s premikajočimi se mesti vzdolž ene črte (*angl. The Single Line Moving Target Traveling Salesman Problem* (SL-MT-TSP)). Najprej bova problem opisali, nato bova predstavili glavne ideje, ki sva jih implementirali v programu in na koncu grafično predstavili še eksperimentalni del projekta.

2 Opis problema

Program za vhodne podatke prejme seznam ciljev oziroma mest, ki jih mora trgovec obiskati. Za vsako mesto je podana lega ob času sprostitve (p), smer gibanja (d; levo ali desno) in čas (r), ob katerem se mesto sprosti (šele tedaj ga lahko trgovec obišče). Poleg tega imamo podano še hitrost, s katero se mesta premikajo, ki je enaka za vsa mesta.

Kot rezultat nam program izračuna **najkrajši čas, ki ga trgovec potrebuje, da obišče vse cilje**, pri čemer je njegova hitrost večja od hitrosti mest, smer njegovega gibanja pa se lahko spreminja. Po lemi, ki je dokazana v članku, na podlagi katerega je zapisan program, je za optimalno rešitev problema možna trgovčeva hitrost v vsakem času enaka eni od vrednosti $-1, 0$ in 1. Hitrost je enaka -1, če je razdalja med zaporedno obiskanima ciljema negativna, 1, če je razdalja pozitivna ter 0, če trgovec predčasno pride na pozicijo, od koder se bo sprostilo naslednje mesto, ki ga želi obiskati - v tem primeru na tej poziciji miruje in čaka sprostitev le-tega mesta. Poleg najhitrejšega časa program vrne tudi vrstni red, po katerem trgovec obišče vsa mesta.

3 Glavne ideje programa

Problema se lotimo tako, da podane cilje razvrstimo v neko ureditev. To lahko naredimo na 2 načina: glede na začetne (ob času 0) pozicije ciljev v naraščajočem vrstnem redu (IPO) ter glede na pozicije ciljev po obdobju stabilizacije (TO; obdobje stabilizacije je obdobje od časa 0 do časa, ki ustreza dvema pogojema: sproščen je bil trenutno zadnji cilj in že se je zgodilo sekanje potencialnih trajektorij dveh ciljev). Gibanje ciljev se v resnici začne že takoj ob času 0, vendar jih trgovec tedaj še ne more prestreči (to lahko naredi od časa sprostitve mesta naprej) – vsa mesta imajo torej ob času 0 neko pozicijo na premici, od koder se že takoj na začetku začnejo gibati. Potencialne trajektorije ciljev so premice, ki prikazujejo gibanje ciljev v odvisnosti od časa neodvisno od gibanja trgovca (dejanska trajektorija pa je segment potencialne trajektorije med časom sprostitve cilja in časom obiska trgovca).

Ko ugotovimo, kakšni sta ureditvi, obe še obrnemo okrog in tako dobimo 4 različne ureditve ($IPO, IPOc, TO, TOc$). Ob vsakem novem obisku mesta v vsaki ureditvi poiščemo mesto, ki je čimbolj na začetku seznama in je še aktivno (ni še bilo obiskano). Tako dobimo C, ki je nabor teh 4 mest. Stanje celotnega sistema ob času obiska nekega mesta je v celoti definirano z dotičnim mestom in omenjenim naborom mest (pri tem predpostavljamo, da je C že posodobljen in ne nakazuje več na pravkar obiskani cilj). Problem rešimo s pomočjo rekurzije, ki se začne v »končnem stanju«, ko so že vsa mesta obiskana, in se po korakih vrača do začetka.

Nabor, ki določa končno stanje, mora vsebovati že obiskane cilje, kar zagotovimo tako, da vanj na vsa 4 mesta zapišemo cilj, ki presega prvotne cilje. Končno stanje dopolnimo še z enim od prvotno zastavljenih mest. Najkrajši čas obiska vseh ciljev izračunamo tako, da za vsakega od ciljev izračunamo vrednost funkcije F za končno stanje, dopolnjeno s tem ciljem. Minimalna od dobljenih vrednosti je naš rezultat ($F*$).

Preden prikaževa implementacijo funkcije f, morava najprej predstaviti še funkcijo g, ki jo f vsebuje. S funkcijo g izračunamo najhitrejši čas, ob katerem lahko obiščemo nek cilj glede na cilj, ki smo ga obiskali nazadnje (pri tem potrebujemo podatek o času, ko smo obiskali zadnji cilj). Funkcija deluje tako, da izračuna razdaljo med zaporednima ciljema in čas, ki ga trgovec potrebuje, da to razdaljo prepotuje. Hitrost trgovca je tu odvisna od predznaka razdalje med ciljema in je enaka 1 ali -1 (kot je pojasnjeno višje v besedilu). Če trgovec na pozicijo cilja pride pred časom sprostitve dotičnega cilja, mora tam počakati na sprostitev (tedaj je njegova hitrost enaka 0), sicer pa lahko cilj prestreže takoj in nadaljuje svojo pot. Funkcija si poleg omenjenega časa zapomni tudi opravljeni odsek poti, kar sva kasneje lahko uporabili ob rekonstruiranju končne poti.

```
def g(self, t, j, i): # najhitrejsi cas obiska cilja i, ce smo nazadnje obiskali cilj j ob casu t
    posi = i.pozicija(t, self.v) # dolocimo poziciji ciljev i in j ob casu t
    posj = 0 if j is None else j.pozicija(t, self.v)
    razlika = posi - posj
    delta = 1 if razlika > 0 else -1 # hitrost gibanja agenta (pozitivna (negativna), ce je razlika med zaporednima ciljema pozitivna (negativna))
    tt = t + razlika / (delta - self.v * i.d) # najmanjsi potreben cas obiska i, dodan k trenutnemu casu t
    if tt >= i.r: # primer, ko trgovec doseze i potem, ko je bil ta ze sproscen
        return (tt, [(t, posj, j), (tt, posi, i)])
    # s trojicami znotraj [] si zapomnimo odseke trgovceve poti (na katerih pozicijah je bil ob casih t in tt in kateri cilj je bil tedaj dosezen)
    else: # ce cilj i v trenutku, ko trgovec doseze njegovo pozicijo, se ni bil sproscen, mora trgovec tam pocakati na sprostitev
        return (i.r, [(t, posj, j), (t + abs(i.p - posj), i.p, None), (i.r, i.p, i)])
    # v zadnjem primeru (srednji element seznama) agent caka na poziciji i.p do casa i.r (v tem primeru je delta = 0)
```

Listing 1: izsek kode - g

```
def f(self, C, i): # minimalni cas, da dosezemo stanje (C, i) - glede na predhodno stanje in predhodnika
    if (C, i) not in self.F: # preverimo, da stanja se nismo poracunali
        kandidati = [] # shranimo vse izracune, potrebovali pa bomo le minimalnega
        CC = self.predhodno_stanje(C, i)
        for l in range(4): # za vsako ureditev predpostavimo, da je predhodnik prisel iz nje
            j = self.predhodnik(l, C, i)
            # ce trenutni seznam ne da kandidata, ga preskocimo
            if j is None:
                continue
            (t, _), _, _ = self.f(CC, j)
            kandidati.append((self.g(t, j, i), l, j))
            # vsak kandidat je dolocen s 3 argumenti:
            # * rezultat funkcije g (najhitrejsi cas obiska cilja i, ce smo nazadnje obiskali cilj j ob casu t),
            # * indeks seznama, iz katerega je prisel naslednji obiskani cilj (l)
```

```
15                  # * predhodnik (j)
16                  # najmanjsega od kandidatov shranimo v F pod kljuc
    (C, i) - pripadajoce stanje
17              self.F[C, i] = min(kandidati) if kandidati else (self.g
    (0, None, i), None, None)
18          return self.F[C, i]
```

Listing 2: izsek kode - f

Funkcija F za vsako stanje poišče minimalni čas, ki ga potrebujemo, da to stanje dosežemo. To naredi tako, da izbere ureditev, iz katere naj bi prišlo zadnje obiskano mesto ter izračuna najhitrejši možen čas obiska cilja, ki ga trenutno želimo obiskati, glede na zadnji obiskani cilj in čas, ko se je ta obisk zgodil. Ta čas pa je odvisen od nabora mest, ki je predhodno trenutnemu in zadnjega obiskanega mesta, ter se spet izračuna s pomočjo funkcije F (tu torej nastopi rekurzija).

Opisana funkcija se izvede za vsako ureditev, minimalna vrednost pa je dejanski rezultat funkcije F. Na ta način smo dobili le najkrajši čas, ki ga trgovec potrebuje za obisk vseh ciljev. Vrstni red ciljev na tej poti pa dobimo tako, da si pri računanju funkcije F zapomnimo dobljena stanja (nabor ciljev in trenutno obiskani cilj) ter čase, ko le-ta stanja dosežemo. Na koncu stanja le še razporedimo glede na čase, izračunane s funkcijo F ter izpišemo v tistih trenutkih obiskane cilje v enakem vrstnem redu.

OPOMBA (včasih so mesta obiskana hitreje kot je planirano oziroma kot izračuna algoritem): Program, ki sva ga napisali, točno izračuna najmanjši čas, ki ga trgovec porabi, da obišče vse cilje. Vrstni red le-teh, ki ga program na koncu vrne pa je pravzaprav le ena od možnih (včasih tudi edina) rešitev. Lahko se zgodi, da so nekateri cilji »pomotoma« obiskani že prej kot to predvidi izračun algoritma. Ko pride do tega, sta mogoči dve situaciji: če je cilj, ki se pojavi na trgovčevi poti, na vrhu vsaj ene od ureditev, obstaja alternativno stanje, ki dominira nad prvotnim stanjem, zato trgovec najprej obišče omenjeni cilj, šele nato nadaljuje svojo pot po prvotnem načrtu. Če pa omenjeni cilj ni na vrhu nobene od ureditev, je sicer naključno obiskan že prej, uradno pa ga v končni rešitvi obiščemo kasneje. Vseeno pa lema (dokazana v izvornem članku) zagotavlja, da to ne vpliva ne na gibanje trgovca in ne na izračunan najkrajši čas. Lastnost ciljev, da bi potencialno lahko bili obiskani predčasno, lahko torej zanemarimo.

4 Generiranje podatkov

Za izvajanje programa sva s pomočjo funkcije generiraj_sezname_d_r_p(a, b, e, f, n) generirali naključne podatke za problem. Funkcija sprejme naslednje vhodne podatke:

- a . . . levo krajišče intervala sprostitvenega časa ciljev
- b . . . desno krajišče intervala sprostitvenega časa ciljev
- e . . . levo krajišče intervala začetnih pozicij ciljev
- f . . . desno krajišče intervala začetnih pozicij ciljev
- n . . . število želenih obravnavanih ciljev

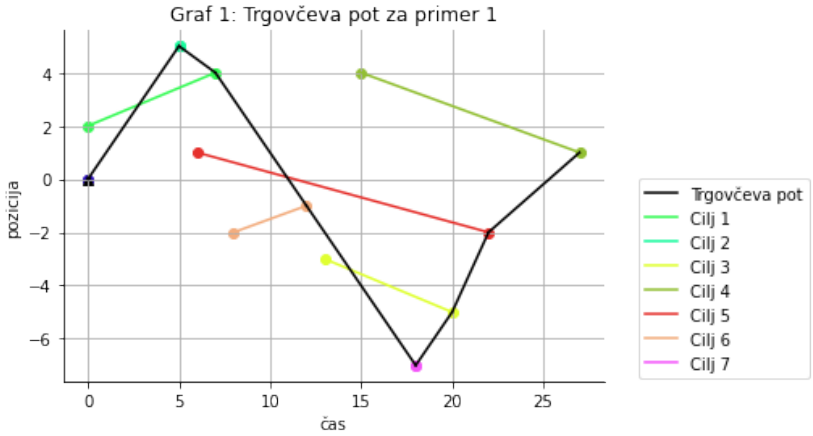
5 Eksperimentalni del

V eksperimentalnem delu projektne naloge sva program poskušali uporabiti na različnih vrstah podatkov. Najprej naju je zanimalo, kako bodo izgledali rezultati, če bodo začetne pozicije ciljev različne in časi sprostitve posameznih ciljev prav tako. Za omenjen primer (primer 1) sva si pogledali tudi rezultate pri različnih hitrostih ciljev. Nadalje so naju zanimali rezultati v primeru, da imajo vsi cilji isto začetno pozicijo in nazadnje še kakšni bodo, če bo čas sprostitve za vsakega izmed ciljev enak 0 . Poleg tega sva proučili tudi kakšni so rezultati in v kolikšnem času so na voljo glede na velikost vhodnih podatkov. Za posamezne primere sva pripravili tudi grafični prikaz trgovčeve poti, ki ga bova tudi opisali v nadaljevanju poročila.

5.1 Primer 1: Trgovčeva pot, kjer so začetne pozicije ciljev časi sprostitve le-teh različni

Tabela 1: Vhodni podatki za primer 1

Cilj j	Cilj 1	Cilj 2	Cilj 3	Cilj 4	Cilj 5	Cilj 6	Cilj 7
r_j	0	5	13	15	6	8	18
p_j	2	5	-3	4	1	-2	-7
d_j	1	1	-1	-1	-1	1	-1



Zgornji graf prikazuje, da trgovec začne svojo pot v izhodišču ob času 0. Nato pot nadaljuje v desno smer in najprej prestreže cilj 2. Potem svojo pot nadaljuje nazaj proti izhodišču in prestreže cilj 1. Pot nadaljuje v isti smeri in po tem, ko prečka izhodišče prestreže cilj 6. V času 18 prestreže cilj 7 nakar ponovno zamenja smer in se ponovno vrača proti izhodišču. Na poti do njega prestreže cilja 3 in 5, ko pa zopet prečka izhodišče, končno prestreže še cilj 2. Za osvojitev vseh ciljev je torej potreboval približno 25.77 časovnih enot. Prikazana pot je planirana trgovčeva pot, ki pa se od dejanske razlikuje. Kot sva nakazali v opombi, trgovec namreč cilja 5 in 3 prestreže že prej, najprej 5 ko prečka izhodišče in nato še cilj 3.

```
# primer uporabe
v1 = 0.3 # hitrost premikanja ciljev
# podatki za cilje: smer gibanja, cas sprostitve, pozicija ob
    sprostitvi
cilji1 = [Cilj(*p) for p in zip([1, 1, -1, -1, -1, 1, -1],
                                [0, 5, 13, 15, 6, 8, 18],
                                [2, 5, -3, 4, 1, -2, -7])]
primer1 = SLMTTSP(cilji1, v1)
resitev1 = primer1.resi()
print(resitev1)
```

Listing 3: textv1 = 0.3

```
(25.769230769230766, [(0, 0, None), (5.0, 5.0, Cilj(1, 5, 5)),
    (6.153846153846153, 3.846153846153846, Cilj(1, 0, 2)),
    (11.076923076923077, -1.076923076923077, Cilj(1, 8, -2)), (18,
    -7.0, Cilj(-1, 18, -7)), (19.923076923076923,
    -5.076923076923077, Cilj(-1, 13, -3)), (21.384615384615383,
    -3.615384615384615, Cilj(-1, 6, 1)), (25.769230769230766,
    2.084615384615385, Cilj(-1, 15, 4))])
```

Listing 4: Rezultati v1 = 0.3

Kot sva že zgoraj zapisali, je v primeru 1, pri hitrosti ciljev 0.3 trgovec vse cilje obiskal v času 25.769230769230. Program v prvi komponenti rešitve vrne čas, ki ga je trgovec potreboval, da je obiskal vse cilje, v drugi pa seznam, ki opisuje njegovo pot. Vsaka komponenta seznama predstavlja čas, pozicijo in cilj sam, pri čemer je seznam urejen po vrstnem redu, v katerem je trgovec obiskal cilje. Nadalje naju je zanimalo, kakšni o rezultati v primeru, da imjo cilji drugačno hitrost. Najprej sva si ogledali primer, ko je hitrost vsakega izmed ciljev enaka v2 = 0.000001. V tem primeru sva dobili naslednje podatke:

```
(26.000008000008002, [(0, 0, None), (5.0, 5.0, Cilj(1, 5, 5)), (15,
     4.0, Cilj(-1, 15, 4)), (16.999983000017, 2.000016999983, Cilj
    (1, 0, 2)), (18.000012000012, 0.999987999988, Cilj(-1, 6, 1)),
    (20.999987000013, -1.999987000013, Cilj(1, 8, -2)),
    (22.000009000009, -3.000009000009, Cilj(-1, 13, -3)),
    (26.000008000008002, -7.000004000009, Cilj(-1, 18, -7))])
```

Listing 5: Rezultati v2 = 0.000001

Vhodni podatki so bili v tem primeru enaki, le hitrost ciljev je bila manjša kot zgoraj. Program nama je rešitev vrnil nekoliko hitreje, vidimo pa lahko, da je rešitev drugačna - zaporedje, v katerem je trgovec obiskal cilje je drugačno, prav tako pa je za osvojitev le-teh potreboval malenkost več časa in sicer 26.000008000008.

Na koncu sva rezultate primerjali še s situacijo, ko je hitrost ciljev enaka $v3 = 0.999999$. Dobili sva naslednje rezultate:

```
(26999999.99916268, [(0, 0, None), (6, 1.0, Cilj(-1, 6, 1)), (13,
    -3.0, Cilj(-1, 13, -3)), (18, -7.0, Cilj(-1, 18, -7)),
    (22.00000350000175, -2.9999964999982494, Cilj(-1, 15, 4)),
    (15000007.999568665, 14999982.999568665, Cilj(1, 8, -2)),
    (25000004.998559773, 24999979.998559773, Cilj(1, 5, 5)),
    (26999999.99916268, 24999981.998554774, Cilj(1, 0, 2))])
```

Listing 6: Rezultati v3 = 0.999999

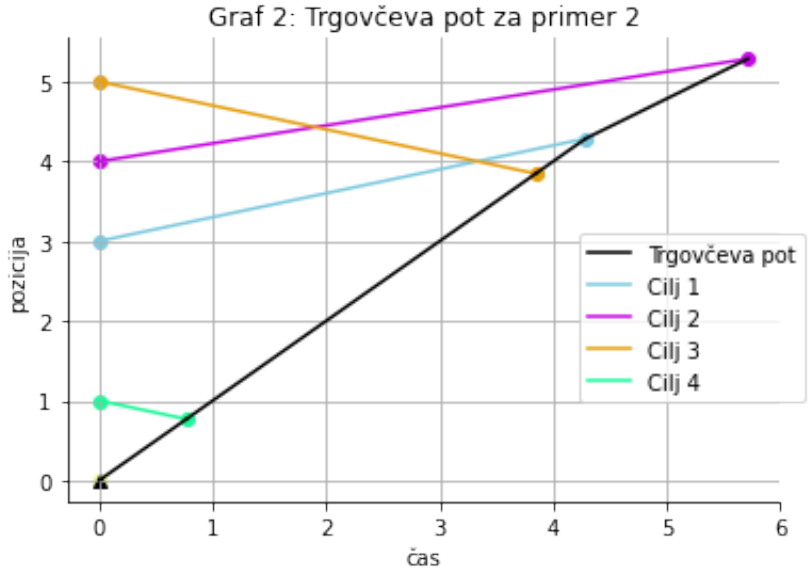
Da sva do njih prišli, je trajalo nekoliko dlje kot v prejšnjih dveh primerih. Razlika je tudi v rezultatih samih. Trgovec v tem primeru za osvojitev vseh ciljev namreč potrebuje več časa in sicer 26999999.99916.

5.2 Primer 2: Trgovčeva pot, kjer so začetne pozicije ciljev različne, časi sprostitve vseh pa so enaki 0

V drugem primeru sva si pogledali, kako bo izgledala trgovčeva pot, če bodo časi sprostitve vseh mest enaki 0, torej se bo vsak cilj sprostil takoj na začetku. Hitrost ciljev je, kot v prvem obravnavanem primeru enaka 0.3.

Tabela 2: Vhodni podatki za primer 2

Cilj j	Cilj 1	Cilj 2	Cilj 3	Cilj 4
r_j	0	0	0	0
p_j	3	4	1	1
d_j	1	1	-1	-1



Iz grafa lahko vidimo, da se vsi cilji začnejo premikati ob času 0. Trgovec bo v tem primeru najprej prestregel cilj 4, nato cilj 3, cilj 1 in nazadnje cilj 2. Kot lahko vidimo spodaj, njegova pot traja 5.714285714285713. Rezultate sva dobili nekoliko hitreje kot v prejšnjih primerih, saj sva v tem primeru opazovali le 4 cilje, medtem ko sva jih prej 7.

```
(5.7142857142857135, [(0, 0, None), (0.7692307692307692,
    0.7692307692307693, Cilj(-1, 0, 1)), (3.846153846153846,
    3.8461538461538463, Cilj(-1, 0, 5)), (4.285714285714285,
    4.285714285714286, Cilj(1, 0, 3)), (5.7142857142857135,
    5.285714285714286, Cilj(1, 0, 4))])
```

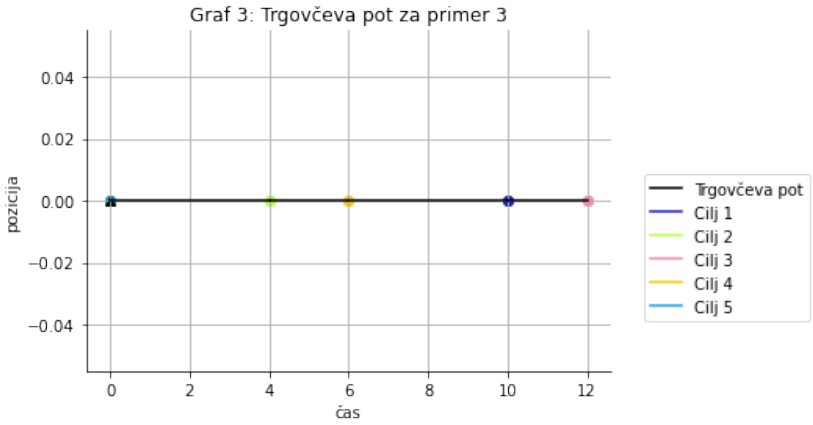
Listing 7: Rezultati v3 = 0.3

5.3 Primer 3: Trgovčeva pot, kjer so začetne pozicije vseh ciljev enake 0, časi sprostitve le-teh pa so različni

V zadnjem primeru, ki ga bova predstavili grafično, sva opazovali množico petih ciljev, katerih začetna pozicija je enaka začetni poziciji trgovca, časi sprostitve ciljev pa so različni. Tudi v tem primeru je hitrost ciljev enaka 0.3.

Tabela 3: Vhodni podatki za primer 3

Cilj j	Cilj 1	Cilj 2	Cilj 3	Cilj 4	Cilj 5
r_j	10	4	12	6	0
p_j	0	0	0	0	0
d_j	1	1	-1	-1	1



Kot bi lahko pričakovali, iz grafa vidimo, da trgovec vsakega izmed ciljev prestreže takoj, ko se le-ta sprosti. Prej kot se cilj sprosti, prej ga bo trgovec prestregel, torej je zaporedje njegove poti definirano z naraščajočim zaporedjem sprostitvenih časov ciljev. Njegova pot se konča, ko se sprosti zadnji cilj in to je v tem primeru ob času 12, kar lahko vidimo tudi iz spodnjih rezultatov.

```
(12, [(0, 0, None), (0.0, 0.0, Cilj(1, 0, 0)), (4, 0.0, Cilj(1, 4,
    0)), (6, 0.0, Cilj(-1, 6, 0)), (10, 0.0, Cilj(1, 10, 0)), (12,
    0, Cilj(-1, 12, 0))])
```

Listing 8: Rezultati v3 = 0.3

Na koncu sva testirali še, kako se program odziva na velikost vhodnih podatkov. V primeru, ko je vsak vhodni seznam vseboval 7 elementov, sva rezultate dobili v slabih 3 sekundah, ko je vseboval 9 elementov pa v 8 sekundah. Nato sva program preizkusili na vhodnih podatkih, ki so vsebovali vsak po 15 elementov in prišli do rezultatov v 130 sekundah. Za rezultate, kjer je bilo vhodnih elementov v vsakem seznamu 35 pa je program potreboval približno pol ure.

6 Literatura

Hassoun M., Shoval S., Simchon E., Yedidsion L. (2019). *The single line moving target traveling salesman problem with release times.* Springer Science+Business Media.