UNIVERZA V LJUBLJANI
FAKULTETA ZA MATEMATIKO IN FIZIKO

Finančna matematika – 1. stopnja

Miha Brešar

Timotej Vesel

Optimizacija s kolonijami mravelj za problem trgovskega potnika

Projekt iz OR pri predmetu Finančni praktikum

Poročilo

Ljubljana, 2019

Kazalo

1. Opis problema

Problem trgovskega potnika je zelo realen problem, ki se velikokrat pojavlja npr. pri trgovskih potnikih, poštarjih, voznikih dostavnih vozil in ljudeh podobnih poklicev. Problem se glasi: "imamo množico mest in razdalj med vsakim parom mest. Zanima nas kakšna je najkrajša oziroma najcenejša pot, ki obišče vsa mesta in se vrne v začetno mesto."

Matematično lahko opišemo *problem trgovskega potnika* na naslednji način. Dana je množica mest $C = \{c_1, c_2, \ldots, c_n\}$. Za vsak par mest c_i, c_j je znana cena povezave od mesta c_i do mesta c_j, ki jo označimo z d_{ij}. Trgovski potnik mora začeti pot v enem od mest, obiskati vsa preostala mesta s seznama ter se vrniti v izhodišče, tako da bo skupna cena poti čim manjša. Z drugimi besedami, poiskati želimo takšno zaporedje mest $(c_{\pi_1}, c_{\pi_2}, \ldots, c_{\pi_n})$ iz C, da bo vrednost izraza $d_{\pi_1,\pi_2} + d_{\pi_2,\pi_3} + \ldots + d_{\pi_{n-1},\pi_n} + d_{\pi_n,\pi_1}$ najmanjša.

Nalogo problema trgovskega potnika zelo naravno predstavimo z grafom, v katerem so mesta vozlišča grafa. Vsako vozlišče grafa povežemo z vsemi drugimi vozlišči, povezavi pa priredimo število, ki je enako ceni poti med mestoma, ki predstavljata krajišče povezave. (Če povezava med dvema mestoma ne obstaja, jo dodamo in ji priredimo ceno ∞. S tem se optimalna rešitev ne spremeni.)

Problem trgovskega potnika je NP-težek problem, torej ne obstajajo algoritmi, ki bi do rešitve prišli v polinomskem času.Zato se bomo pri reševanju "velikih" nalog tega problema morali zadovoljiti s približkom optimalne rešitve, saj je eksaktno reševanje za te primere računsko prezahtevno. V ta namen bomo uporabili metahevristiko kolonije mravelj, ki izračuna približek optimalne rešitve naloge Problema trgovskega potnika v polinomskem času.

2. Optimizacija s kolonijami mravelj

Optimizacija s kolonijami mravelj je zasnovana na dejstvu, da so v naravi mravlje sposobne poiskati najkrajšo pot od vira hrane do mravljišča brez uporabe vizualnih informacij. Za medsebojno komunikacijo mravlje uporabljajo feromon, ki ga odlagajo med hojo. Poti, ki so bolj obiskane, imajo zato večjo količino feromona. Vsaka mravlja daje prednost sledenju smerem, ki so bogatejše s feromonom.

Ob dani nalogi problema trgovsega potnika z n mesti in razdaljami oz. cenami d_{ij} so umetne mravlje naključno razporejene med temi n mesti. Vsaka mravlja bo nato izbrala naslednje mesto, ki ga bo obiskala, glede na količino preostalega feromona na poteh, ki vodijo iz mesta v katerem se trenutno nahaja (glej primer v kratki predstavitvi). Obstajata pa dve veliki razliki med umetnimi in pravimi mravljami. Prva razlika je,

da imajo umetne mravlje spomin. To pomeni, da si lahko zapomnijo mesta, ki so jih že obiskala in jih zato ne bodo več obiskala. Drug razlika pa je, da umetne mravlje niso popolnoma "slepe". Poznajo razdalje med dvema mestoma in zato preferirajo izbiro bližnjih mest iz njhove trenutne pozicije. Verjetnost, da mravlja k, ki se nahaja v mestu i izbere mesto j lahko zapišemo kot

$$P_{ij}^k = \begin{cases} \frac{[\tau_{ij}]^\alpha [\eta_{ij}]^\beta}{\sum_{s \in neobiskana_k} [\tau_{ij}]^\alpha [\eta_{ij}]^\beta} & j \in neobiskana_k, \\ 0 & \text{sicer.} \end{cases} \tag{1}$$

kjer je τ_{ij} količina feromona na poti med mestoma i in j, α parameter, ki določa vpliv τ_{ij}, η_{ij} vidljivost mesta j iz mesta i in je vedno nastavljena na $1/d_{ij}$, β parameter, ki regulira vpliv η_{ij} in $neobiskana_k$ množica mest, ki jih mravlja k še ni obiskala.

Na začetku m mravelj naključno razporedimo med n mest. Potem se vsaka mravlja odloči katero je naslednje mesto, ki ga bo obiskala, glede na na verjetnost P_{ij}^k, ki je podana z enačbo (1). Po n iteracijah procesa vse mravlje zaključijo obhod.

Lokalna posodobitev feromona se izvede za vsako povezavo po kateri potuje mravlja, da se prepreči, da bi naslednje mravlje šle po isti poti

$$\tau_{ij} = (1 - \sigma) \cdot \tau_{ij}(t) + \sigma \cdot \tau_{ij}^0 \tag{2}$$

kjer je τ_{ij} količina feromona na povezavi (i, j), σ parameter lokalnega feromona in τ_{ij}^0 začetna vrednost feromona.

Smiselno je, da bi mravlje s krajšimi obhodi na poti pustile več feromona kot tiste, katerih obhodi imajo večjo dolžino (želimo, da dobijo obhodi z manjšo dolžino večjo prioriteto). Zato po koncu vsake iteracije vsem povezavam, ki so vsebovane v trenutni najboljši rešitvi dodamo količino feromona Q/L, pri čemer je Q konstanta, L pa dolžina trenutnega najkrajega obhoda. Po drugi strani pa feromon sčasoma izhlapeva. Zaradi tega lahko pravilo za posodobitev τ_{ij} zapišemo kot

$$\tau_{ij}(t + 1) = (1 - \rho) \cdot \tau_{ij}(t) + \rho \cdot \Delta\tau_{ij} \tag{3}$$

$$\Delta\tau_{ij} = \begin{cases} Q/L & \text{povezava } (i, j) \text{ je vsebovana v trenutni najboljši rešitvi,} \\ 0 & \text{sicer.} \end{cases} \tag{4}$$

kjer je t števec iteracij, τ_{ij} količina feromona na povezavi (i, j), $\rho \in [0, 1]$ paramter, ki regulira zmanjševanje (izhlapevanje) feromona in $\Delta\tau_{ij}$ povečanje količine feromona na povezavi (i, j).

Najkrajsega izmed obhodov, ki so ga opravile mravlje si shranimo, če je manjši od do zdaj najdenega minimuma. Po posodobitvi količine feromona na poteh, sledi iteracija $t + 1$. Ta postopek ponavljamo, dokler ne dosežemo nekega zaustavitvenega pogoja, navadno je to fiksno naravno število, ki omejuje število iteracij. Spodaj je prikazana psevdokoda algoritma.

Naj na tem mestu omenimo, da obstaja več različic optimizacije s kolonijami mravelj. Mi smo implementirali sistem kolonije mravelj (Ant colony system), ki se od ostalih razlikuje predvsem po tem, da se feronom na koncu vsake iteracije dodaja le na povezave, kjer je bila najdena do sedaj najboljša pot. S tem se časovna zahtevnost globalnega posodabljanja feronoma zmanjša na $\mathcal{O}(n)$. Poleg tega je prednost te različice algoritma še ta, da ima lokalno posodabljanje feromona. Torej mravlje zmanjšajo količino feromona na poti po kateri gredo, kar zmanjša ponavljanje poti oziroma so zgodnja mesta v poti ene mravlje pozneje raziskana v poteh drugih mravelj. Povedano drugače, učinek lokalnega posodabljanja je, da se zaželenost povezav spreminja: vsakič ko mravlja uporabi povezavo, ta postane malo manj zaželena (ker izgubi nekaj svojega feromona). Na ta način bodo mravlje bolje izkoristile informacijo o feromonu, saj bi brez lokalnega posodabljanja vse mravlje iskale v ozki soseščini najboljše prejšnje poti.

Algoritem 1: Ant colony za TSP

Vhod: *Množica mest* $C = \{c_1, c_2, \ldots, c_n\}$*, cene* d_{ij} *za vsak par* c_i, c_j *in ostali potrebni parametri* $(\tau_{ij}, \alpha, \beta, \ldots)$*.*
Izhod: *Zaporedje mest* $D = (c_{\pi_1}, c_{\pi_2}, \ldots, c_{\pi_n})$ *v najkrajšem obhodu in njegova dolžina.*

```
begin
    for t = 1 to število iteracij do
        for k = 1 to m do
            while mravlja k zaključi obhod do
                izberi mesto j, ki bo obiskano naslednje z
                  verjetnostjo P_ij podano z enačbo (1)
                lokalna posodobitev feromona z enačbo (2)
            end
        end
        posodobi količino feromona glede na enačbi (3) in (4)
    end
end
```

3. Iskanje in generiranje podatkov

Algoritem je napisan tako, da potrebujemo kvadratno matriko uteži velikosti n, katere elementi so dolžine oziroma cene med posameznima vozliščema.

Prvotna verzija algoritma omogoča generiranje grafov s povsem naključnimi utežmi. Pri tem algoritmu moramo določiti samo število mest in zgornjo mejo za uteži. Pri takšnih grafih seveda lahko med seboj primerjamo različne vrednosti parametrov, vendar pa ne vemo kako dobre so naše rešitev. Želimo pa si, da bi lahko naše rešitve primerjali

z rešitvami dobljenimi z drugimi algoritmi. Zaradi tega smo se odločili na spletu poiskati nekaj "znanih" nalog problema trgovskega potnika in naše rešitve primerjati z najboljšimi znanimi rešitvami teh nalog.
Ločimo *simetrični* in *nesimetrični* TSP. Pri simetričnem je razdalja med dvema mestoma enaka v obe smeri, torej imamo neusmerjen graf. Pri nesimetričnem TSP pa se lahko zgodi, da pot med dvema vozliščema ne obstaja v obeh smereh ali pa razdalji v obeh smereh nista enaki.

Pri simetričnih problemih trgovskega potnika je bilo največ uteži v evklidski obliki, t.j. za vsako vozlišče poznamo koordinati x in y v ravnini. Zaradi tega smo potrebovali funkcijo, ki izračuna (evklidske) razdalje med dvema vozliščema in te razdalje zapiše v matriko uteži.

Pri nekaterih simetričnih nalogah, ki smo jih obravnavali, pa so bile uteži eksplicitno podane v obliki zgornje/spodnje trikotne matrike. Zaradi tega smo mogli napisati še funkcijo, ki nam je to obliko zapisa pretvorila v simetrično kvadratno matriko velikosti n.

Odločili pa smo se tudi za obravnavo nekaj nalog nesimetričnega problema trgovskega potnika. Pri nekaterih nalogah so podatki že bili v kvadratni matriki velikosti n in smo jih morali samo pretvoriti iz tekstovne datoteke v matriko. Pri nekaterih nalogah pa so bili podatki zapisani v ne-kvadratnih matrikah (vrstica matrike je bila razpisana v več vrstic). Zaradi tega smo potrebovali še funkcijo, ki nam je to obliko zapisa pretvorila v kvadratno matriko velikosti n.

4. Opis eksperimentov

4.1. **Simetrične instance.** Prva naloga, ki smo jo testirali je bil evklidski graf na 51 vozliščih (*eil51*). Začetni parametri so bili:

- $\alpha = 2.5$ (vpliv vidljivosti)
- $\beta = 7$ (vpliv feromona)
- $c = 0.1$ (začetni feromon)
- $\sigma = 0.1$ (parameter lokalnega feromona)
- $\rho = 0.08$ (parameter izhlapevanja feromona)
- $q_0 = 0.9$ (požrešnost)
- $m = 10$ (število mravelj)
- $q = 0.01$ (količina feromona, ki ga izloči mravlja)

Vendar smo s temi parametri takoj dobili rešitev, ki je bila enaka najboljši poznani rešitvi. Zato smo se odločili, da bomo vpliv parametrov raje preizkušali na nekoliko večjem grafu.

Izbrali smo si evklidski graf na 70 vozliščih (*st70*). Z zgornjimi paramteri smo bili precej oddaljeni od najboljše poznane rešitve. Odločili smo se za spreminjanje parametrov α in β. Ob zmanjšanju parametra β na 2.5 smo dobili še precej slabšo rešitev. Zato smo β raje povečali na 7, vendar je bila rešitev samo nekoli boljša od tiste pri začetnih parametrih.

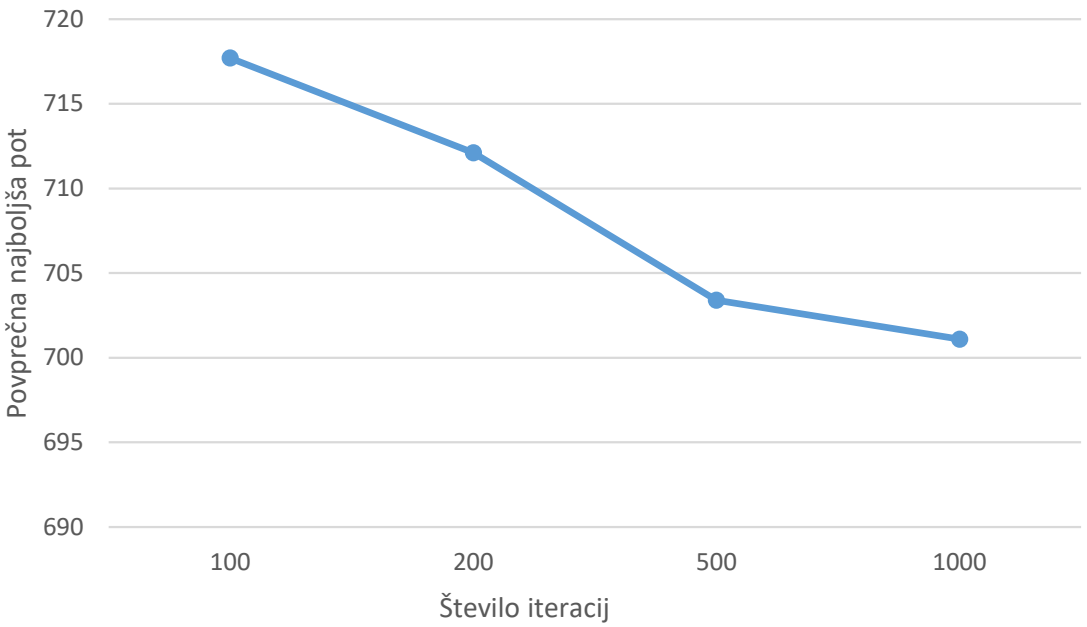
Ker algoritem temelji na verjetnostih, obstaja možnost, da imamo srečo in v eni izmed prvih ponovitev dobimo zelo dober rezultat. Ker smo pri zgoraj opisanih algoritem ponovili samo desetkrat, se nam je zdelo smiselno, da bi nekoliko povečali število izvedb algoritma in si poleg tega ogledali še povprečje vseh najboljših izračunanih poti. Zato smo zapisali funkcijo *average*, ki nam izračuna to povprečje ter hkrati vrne še najkrašo pot in njeno dolžino.

Pri naslednjih testiranjih smo še povečevali parameter β vendar smo ugotovili, da povečanje prek 7 ni smiselno (za $\beta > 10$ so rezultati že precej slabši). Nato smo povečali še α in prišli do ugotovitve, da so rezultati precej boljši za $\alpha = 2.5$ in $\alpha = 3$, za večje α (npr. že $\alpha = 5$) pa so precej slabši.

Dalje smo povečali še količino feromona (q), ki ga izloči mravlja na 0.1, 0.15, 0.4, 0.5 (količina feromona na povezavah na trenutni najkrajši poti se poveča za q/(dolžina trenutne najkrajše poti) in dobivali nekoliko boljše rezultate kot pred tem. Nato smo spreminjali še parametra ρ in σ ter v povprečju najboljši rezultat dosegli pri $\rho = 0.08, \sigma = 0.1, q = 0.4, \alpha = 2.5, \beta = 7$. Najboljši rezultat, 677 (najboljša znana rešitev je 675) pa smo dobili pri pri $q = 0.15$. Vendar kot že rečeno je možno, da bi, če bi algoritem zagnali večkrat dobili tudi 675.

Povečati smo poskusili tudi število mravelj, saj več mravelj povzroča hitrejše izhlapevanje feronoma, kar lahko povzroči hitrejšo konvergenco. Vendar pa lahko zaradi tega pademo v enega od prvih lokalnih minimumov, ki jih najdemo, kar pa seveda ni dobro. Vendar rezultati niso bili nič boljši, za izračun pa je algoritem potreboval več časa (linearno povečanje, npr. če podvojimo število mravelj, se podvoji čas računanja). Izkaže se, da je ravno 10 optimalno število mravelj za Sistem kolonije mravelj.

Pomemben vpliv na kakovost rezultatov pa ima število iteracij. Kakšno je najboljše število iteracij smo preizkusili na dveh primerih. Na naši testni instanci *st70* in nato še na instanci na 51 vozliščih *eil51*. Preizkusili smo 100, 200, 500 in 1000 iteracij. Prišli smo do ugotovitve, da algoritem za 500 iteracij deluje precej bolje od 100 in 200 iteracij. Z nadaljnim povečanjem števila iteracij pa se izboljševanje rezultatov manjša. Za 1000 iteracij so rezultati že skoraj zanemarljivo malo boljši od rezultatov pri 500 iteracijah. Je pa trajanje izvedbe algortima dvakrat daljše, zato se nam povečevanje iteracij preko 1000 ni zdelo smiselno. Na spodnji sliki je prikazan graf povprečne najboljše poti pri desetih izvedba algoritma za *st70*.



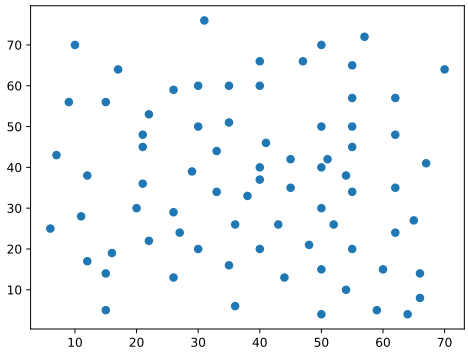
Slika 1. Povprečna najkrajša pot v desetih izvedbah algoritma za instanco *st70*.

Nato smo testiranje parametrov opravili še na dveh grafih ter dobili podobne rezultate. Potem pa smo uspešnost teh parametrov preizkusili še na nekaj evklidskih grafih:

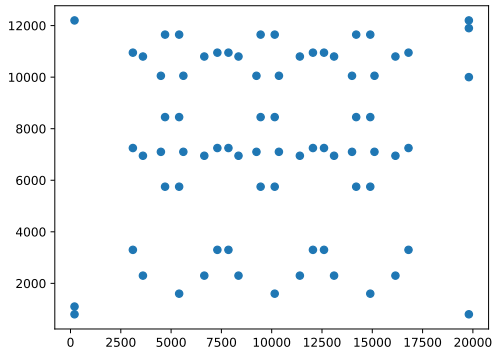
- 131 vozlišč (*xqf131*): 585 (najboljša znana rešitev je 564), relativna napaka = 3.7%
- 76 vozlišč (*pr76*): 109700 (108159), relativna napaka =1.4 %
- 76 vozlišč (*eil76*): 538 (538), relativna napaka = 0 %
- 130 vozlišč (*ch130*): 6226 (6110), relativna napaka = 1.8 %
- 99 vozlišč (*rat99*): 1212 (1211), relativna napaka = 0.08 %
- 195 vozlišč (*rat195*): 2343 (2323), relativna napaka = 0.86 %
- 100 vozlišč (*rd100*): 7950 (7910), relativna napaka = 0.5 %
- 280 vozlišč (*a280*): 2642 (2579), relativna napaka = 2.44 %

Za simetrične grafe, kjer imamo podano zgornje/spodnje trikotno matriko uteži so rezultati seveda podobni, saj imamo pri obeh načinih simetrično matriko uteži.

Opazimo lahko, da se relativni napaki pri dveh nalogah s 76 vozlišči precej razlikujeta. Podobno lahko opazimo tudi na primer pri nalogah z 131 in 195 vozlišči, kjer smo pri nalogi z 195 vozlišči dobili boljšo rešitev kot pri nalogi z 131 vozlišči. Ker so to evklidski grafi, se nam je zdelo smiselno, da si ogledamo položaj vozlišč na koordinatnem sistemu.



Slika 2. Naloga na 76 vozliščih *eil76*



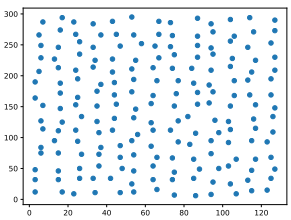
Slika 3. Naloga na 76 vozliščih *pr76*

Opazimo lahko, da so na prvi sliki vozlišča precej bolj skupaj, na drugi pa so precej bolj narazen, poleg tega pa imamo še nekaj "osamelcev", ki so precej oddaljeni od preostalih vozlišč.

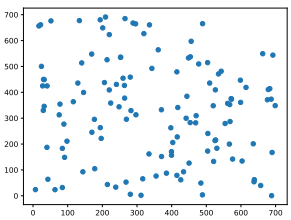
Tudi na spodnjih slikah vidimo podobno. Vozlišča na prvi sliki so precej bolj enakomerno porazdeljena (so bolj skupaj), prav tako pa ni nobenega večjega "osamelca".

Torej je očitno, da poleg števila vozlišč na učinkovitost algoritma vpliva tudi postavitev vozlišč v evklidski ravnini.

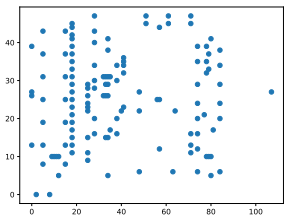
Kot smo že omenili v prejšnjih odstavkih, imamo večjo verjetnost, da dobimo dober rezultat, če algoritem izvedemo večkrat. Še posebej se to pozna pri večjih problemih (pri zelo majhih problemih je precej verjetno, da bomo zelo hitro dobili optimalno rešitev). Vendar pa nam to omejuje časovna zahtevnost algoritma, ki je $\mathcal{O}(n^2)$, kjer je n število mest. Na primer za $n = 50$ potrebuje algoritem približno 50 sekund za $n = 100$ pa že kar 180 sekund.



Slika 4. *rat195*



Slika 5. *ch130*



Slika 6. *xqf131*

4.2. **Nesimetrične instance.** Algoritem smo tudi priredili za nesimetrične instance in ga preizkusili na nekaj le-teh. Rezultati so nekoliko slabši kot pri simetričnih instancah, saj naprimer že pri instanci na 35 vozliščih ne dobimo rezultatov, ki bi bili enaki najboljšim do

sedaj poznanim. Pri simetričnih nalogah bi za "samo" 35 vozlišč ponavadi dobili precej boljše rezultate. Vendar pa je to pričakovano, saj je reševanje naloge TSP na nesimetričnem grafu precej težji problem kot na simetričnem.

5. Zaokrožitvene napake

Pri implementaciji algoritma se soočimo tudi z numeričnimi problemi. Pri računanju s plavajočo vejico se namreč lahko pojavijo zaokrožitvene napake, ki lahko, če smo nepazljivi, usodno vplivajo na rezultat. Problem nastane, ko zaradi zmanjšanja feronoma pride do podkoračitve, kar povrzroči probleme pri izbiranju naslednjega vozlišča, saj so vse verjetnosti enake 0. V Pythonu lahko to rešimo s pomočjo ukazov "Try"in "Except", kjer "Try"deluje kot "if", pri pogoju da ne pride do podkoračitve in "Except"kot "else". V primeru podkoračitve program izbere prvo neobiskano vozlišče in sporoči, da je prišlo do podkoračitve. To je za nas signal, da je potrebno prilagoditi vhodne parametre, saj algoritem namreč ne deluje več pravilno. Do tega lahko pride, če želimo uporabiti enake količine feronoma na mnogo večjem problemu, brez da bi upoštevali velikost problema. Podobna težava nastopi, če želimo preveč povečati parameter, ki uravnava vpliv feronoma. To rešimo tako, da zmanjšamo vrednost parametra, ki uravnava vpliv razdalje, kar da podoben učinek. Dobra stran je, da to teh težav pride le pri zelo velikih grafih in parametrih, ki tudi pri manjših grafih ne dajo dobrih rezultatov.

6. Literatura

[1] Jinhui Yang, Xiaohu Shi, Maurizio Marchese , Yanchun Liang: An ant colony optimization method for generalized TSP problem.

[2] Jason Brownlee: Clever Algorithms: Nature-Inspired Programming Recipes.

[3] Marco Dorigo and Thomas Stützle: Ant Colony Optimization. Mit Press. 2004.

[4] Knjižica TSPLIB. Dostopno na:
http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsplib.html

[5] TSP Test Data. Dostopno na:
http://www.math.uwaterloo.ca/tsp/data/index.html