Pomoč za trgovskega potnika

Aleksander Vesel

Trgovski potnik ima problem

Trgovski potniki, poštarji, vozniki dostavnih vozil, kurirji in ljudje podobnih poklicev nimajo prav lahkega dela. Pred delom dobijo seznam krajev oz. naslovov, ki jih morajo obiskati, ter se vrniti v izhodišče. Pomembno je, da obiščejo vse kraje s seznama in da za obisk porabijo čim manj sredstev, predvsem časa in goriva. Izkaže se, da je pravilna izbira vrstnega reda obiskanih krajev zelo pomembna, saj lahko z nerodno izbiro bistveno povečamo stroške poti.

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 $d_{i,j}$. 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. Zaporedju mest, ki minimizira zgornji izraz, bomo rekli tudi *najcenejša rešitev*.

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.

Kot primer si poglejmo množico mest $C = \{c_1, c_2, c_3, c_4, c_5\}$, cene povezav med njimi pa so $d_{1,2} = d_{2,1} = 132$, $d_{1,3} = d_{3,1} = 308$, $d_{1,4} = d_{4,1} = 68$, $d_{1,5} = d_{5,1} = 233$, $d_{2,3} = d_{3,2} = 180$, $d_{2,4} = d_{4,2} = 66$, $d_{2,5} = d_{5,2} = 114$, $d_{3,4} = d_{4,3} = 240$, $d_{3,5} = d_{5,3} = 80$ in $d_{4,5} = d_{5,4} = 167$. Primer je predstavljen na sliki 1.

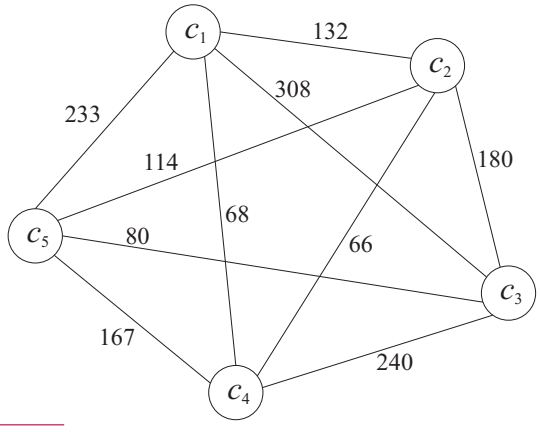
Hitro opazimo, da je cena povezave od mesta c_i do mesta c_j enaka ceni povezave od mesta c_j do mesta c_i, velja torej $d_{i,j} = d_{j,i}$ za vsak par mest c_i in c_j. Omejitev je precej naravna, saj je načeloma dolžina poti med dvema mestoma enaka v obeh smereh. Ker je v praksi pogosto cena povezave sorazmerna dolžini razdalji med mestoma, je v takih primerih zato tudi cena povezave enaka v obe smeri. Problem s to lastnostjo imenujemo *simetrični problem trgovskega potnika*.

Malo težje je opaziti, da v primeru s slike 1 za cene velja *trikotniška neenakost*. S tem povemo, da je cena povezave od mesta c_j do mesta c_i vedno manjša ali enaka vsoti cen povezav od mesta c_j do mesta c_i preko mesta c_k. Z drugimi besedami, za poljubno trojico mest c_i, c_j, c_k velja $d_{i,j} \leq d_{i,k} + d_{k,j}$. Tudi ta omejitev je naravna, saj je dolžina direktne poti med dvema mestoma običajno manjša od dolžine poti, pri kateri naredimo ovinek preko tretjega mesta.

Naivni algoritem

Spomnimo se, da želimo poiskati najcenejšo krožno pot, to je pot, ki se začne v nekem mestu, gre skozi vsa preostala mesta in se zaključi v izhodišču. Hitro opazimo, da je vseeno, v katerem mestu začnemo krožno pot. Brez izgube za splošnost bomo zato vedno začenjali v c_1.

Različnih krožnih poti je veliko, njihove cene pa zelo različne. Če npr. mesta iz slike 1 obiščemo glede na naraščajoče indekse, dobimo krožno pot $(c_1, c_2, c_3, c_4, c_5)$ s ceno 952. Že majhna sprememba v zaporedju lahko povzroči bistveno spremembo cene. Če v zgornjem zaporedju c_2 premaknemo na



SLIKA 1.
Primer problema trgovskega potnika.

konec, dobimo krožno pot $(c_1, c_3, c_4, c_5, c_2)$ s ceno 961, če pa na konec zaporedja premaknemo c_3, dobimo krožno pot $(c_1, c_2, c_4, c_5, c_3)$ s ceno 753. Tudi najcenejšo pot za ta primer lahko hitro najdemo. Ker bomo vedno začeli v mestu c_1, moramo pravilno razporediti preostala štiri mesta. Poiskati moramo vse ureditve zaporedja c_2, c_3, c_4, c_5 in izračunati cene pripadajočih krožnih obhodov. Z drugimi besedami, potrebno je poiskati in ovrednotiti vse permutacije zaporedja s štirimi elementi. Teh je natanko $4! = 24$, zato jih lahko razmeroma enostavno preverimo tudi brez računalnika. Najcenejši obhod s ceno 627 nam tako da zaporedje $(c_1, c_2, c_3, c_5, c_4)$ (glej levo stran slike 2).

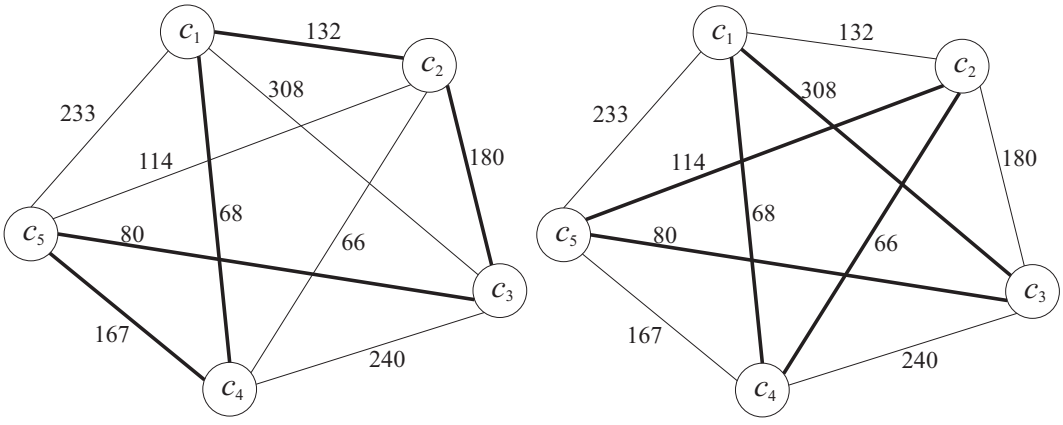
Tudi v splošnem primeru lahko naredimo podobno kot zgoraj: potrebno je samo poiskati vse permutacije zaporedja $c_2, c_3, \ldots, c_n$ in izračunati ceno pripadajočega krožnega obhoda. Opisani postopek zaradi preprostosti imenujemo tudi *naivni algoritem* in ga brez težav zapišemo v programskem jeziku. Ker so računalniki in računalnikom podobne naprave danes zelo vsakdanja stvar, si ni težko zamisliti, da bi takšen program namestili na računalniško tablico ali pametni telefon. Program bi omogočil trgovskemu potniku vnos podatkov o mestih in cenah med njimi ter mu izračunal krožno pot z najmanjšo ceno.

Ali bomo s takim programom v resnici lahko pomagali trgovskemu potniku? V nekaterih primerih že, v vseh pa še zdaleč ne. Spomnimo se, da je vseh permutacij zaporedja $c_2, c_3, \ldots, c_n$ natanko $(n-1)!$. Težava je zato v tem, da število krožnih obhodov, ki jih mora pregledati program, glede na n izredno hitro narašča. Že pri seznamu s štirinajstimi mesti število pregledanih zaporedij presega šest milijard, pri $n = 42$ pa dobimo število krožnih obhodov, ki je večje od skupnega števila vseh atomov na Zemlji!

V realnem svetu to pomeni, da je problem lepo rešljiv za primere, ko je mest na seznamu malo. Ko gre za trgovskega potnika, ki potuje po Sloveniji, je to seveda čisto realna predpostavka. Drugače pa je, ko bi radi pomagali poštarju pri raznosu pošiljk znotraj večjega mesta, saj lahko pričakujemo, da je na njegovem seznamu več deset naslovov. V tem primeru še tako hiter računalnik ne bo pomagal, saj bi bil čas računanja programa mnogo prevelik.

Če hiter računalnik ne pomaga, se je seveda smiselno vprašati, ali bi mogoče pomagal manj naiven in zato hitrejši algoritem. Hitrejši algoritmi v resnici obstajajo, a niso bistveno hitrejši od naivnega algoritma. V praksi to pomeni, da je možno z zmogljivim namiznim računalnikom v smiselnem času rešiti problem za 16 ali 17 mest. Reševanje problema za večje število mest pa bi se hitro zavleklo v več mesecev, let ali celo stoletij, če bi le bilo število mest dovolj veliko.

Problem trgovskega potnika spada med tako imenovane *NP-težke probleme*. Zelo poenostavljeno povedano s tem izrazom označimo probleme, ki jih ne znamo rešiti s hitrimi algoritmi, torej z algoritmi, ki bi omogočali izračun rešitve v sprejemljivem času tudi za večje število vhodnih podatkov. Seznam težkih problemov je zelo dolg, med zelo znane in pro-



SLIKA 2.
Najcenejša rešitev (levo) in rešitev pridobljena z algoritmom najbližja točka (desno).

učevane probleme s tega seznama tako npr. spada tudi problem izdelave urnika.

Aproksimacija

Glede na zgoraj povedano bi lahko pomislili, da trgovskemu potniku, ki mora obiskati večje število mest, sploh ne moremo pomagati. Na srečo zadeva ni popolnoma brezupna, le svoje želje mora trgovski potnik nekoliko prilagoditi. Namesto da bi se trudil z iskanjem najcenejše rešitve, se bo moral zadovoljiti z obhodom, ki zelo verjetno ne bo najcenejši, bo pa izračunan dovolj hitro. Tako izračunanemu zaporedju mest bomo rekli *približna rešitev*.

Obstaja veliko pristopov, kako hitro izračunati približno rešitev problema trgovskega potnika ali kakega drugega težkega problema. Zelo zanimivi so postopki, pri katerih znamo oceniti, za koliko se bo v najslabšem primeru izračunana približna rešitev razlikovala od najcenejše. Algoritme s to lastnostjo imenujemo *aproksimacijski algoritmi*. Posebej zaželjeni so seveda aproksimacijski algoritmi, pri katerih se izračunana približna rešitev preveč ne razlikuje od najcenejše.

Algoritem z zgornjo lastnostjo za splošni problem trgovskega potnika žal ne obstaja, obstajo pa aproksimacijski algoritmi za problem trgovskega potnika, ki zadošča trikotniški neenakosti. Tukaj bomo predstavili algoritem *najbližja točka*.

Algoritem postopoma gradi krožno pot D, ki najprej obsega samo mesto c_1. V nadaljevanju na posameznem koraku doda v krožno pot tisto mesto, iz katerega med vsemi mesti, ki še niso uvrščena v trenutno krožno pot, vodi najcenejša pot do nekega mesta, ki je že na trenutni krožni poti. Algoritem 1 opisuje postopek še formalno.

Algoritem 1: Najbližja točka

Vhod: *Množica mest* $C = \{c_1, c_2, \ldots, c_n\}$, *cene* $d_{i,j}$ *za vsak par* c_i, c_j.
Izhod: *Zaporedje mest* $D = (c_{\pi_1}, c_{\pi_2}, \ldots, c_{\pi_n})$.
begin
  $D := (c_1)$;
  for $i := 2$ **to** n **do**
    x naj bo mesto izven D, iz katerega vodi najcenejša pot do nekega mesta y v D;
    Dodaj x v D takoj za y;
  end
end

Predstavimo potek algoritma za primer s slike 1. Pred vstopom v zanko se v zaporedje D vstavi mesto c_1. Algoritem nato vstopi v zanko in med preostalimi mesti c_2, c_3, c_4, c_5 išče tisto, iz katerega vodi najcenejša pot do c_1. Očitno je to c_4, iz katerega pridemo v c_1 za ceno 68. Podobno se zgodi na naslednjem koraku. Algoritem izmed mest s seznama c_2, c_3, c_5 išče tisto, iz katerega vodi najcenejša pot do c_1 ali c_4. Do c_1 pridemo najceneje iz c_2 za ceno 132, do c_4 pa prav tako iz c_2 za ceno 66. Zato v D dodamo c_2 takoj za c_4. Na podoben način ugotovimo, da najceneje v zaporedje (c_1, c_4, c_2) dodamo c_5 takoj za c_2. Na zadnjem koraku ostane c_3, ki ga najceneje priključimo v obhod za c_5. Podrobneje je potek algoritma predstavljen v tabeli 1.

i	x	y	D	mesta izven D
			(c_1)	c_2, c_3, c_4, c_5
2	c_4	c_1	(c_1, c_4)	c_2, c_3, c_5
3	c_2	c_4	(c_1, c_4, c_2)	c_3, c_5
4	c_5	c_2	(c_1, c_4, c_2, c_5)	c_3
5	c_3	c_5	$(c_1, c_4, c_2, c_5, c_3)$	

TABELA 1.
Potek izračuna algoritma za primer s slike 1.

Izračunano zaporedje predstavlja krožno pot s ceno 636, ki je predstavljena na desni strani slike 2, dobljena rešitev pa je le za dober odstotek slabša od najcenejše krožne poti. V splošnem sicer ne moremo vedno pričakovati tako dobrega približka, dokazano pa je, da je vrednost izračunane rešitve tudi v najslabšem primeru največ dvakratnik najcenejše rešitve. Za konec povejmo, da je znanih še nekaj aproksimacijskih algoritmov za rešitev problema trgovskega potnika, ki zadošča trikotniški neenakosti. Najboljši med njimi je Christofidesov algoritem, s katerim lahko izračunamo približno rešitev, ki se za največ 50 odstotkov razlikuje od najcenejše.

Literatura

[1] T. H. Cormen, C. E. Leiserson in R. L. Rivest, *Introduction to algorithms*, The MIT Press, 2001.

×××