Univerza v Ljubljani
Fakulteta za matematiko in fiziko

Matematika – računalništvo z matematiko (UNI)

Mirjam Kolar

TOP 10

Diplomsko delo

Ljubljana, 2010

Zahvala

Zahvalila bi se vsem, ki ste me skozi vsa ta leta spodbujali in bili z mano. Hvala mentorju za pomoč pri izdelavi diplomskega dela, ostalim za vse nasvete, turistični organizaciji DISKO d.o.o. za idejo teme (upam, da bo projekt tudi uspešno zaživel) in podjetju Premisa d.o.o. za vse proste mesece.

Kazalo

Program dela

V diplomskem delu v matematični obliki prikažite različico problema razporejanja skupin potnikov, s katerim se lahko srečajo v turistični agenciji. Obravnavajte računsko zahtevnost problema, zapišite ga v obliki celoštevilskega linearnega programa ter predlagajte nekaj postopkov za njegovo reševanje (npr. sestopanje, dinamično programiranje, lokalna optimizacija ipd.). Izbrane postopke sprogramirajte in preizkusite ter ovrednotite dobljene rezultate.

Ljubljana, september 2009

izred. prof. dr. Martin Juvan

Povzetek

V diplomskem delu poskušamo najti čim bolj tako časovno kot tudi prostorsko učinkovit algoritem za razporejanje skupin potnikov, ki potujejo med več kraji.

Pri iskanju učinkovitega algoritma si pomagamo z različnimi metodami. Problem poskusimo rešiti s pomočjo sestopanja in s pomočjo dinamičnega programiranja. Ker pa se oba algoritma z večanjem števila skupin in števila krajev izkažeta za ne dovolj dobra, poskusimo tudi, koliko se približamo optimalni rešitvi s pomočjo 2-zamen. Poleg tega poskusimo problem zapisati kot celoštevilski linearni program.

Delo obsega štiri poglavja. V uvodu (prvo poglavje) je predstavljena vsebina diplomskega dela. V drugem poglavju natančno opišemo problem in dodamo enostaven primer, da si problem lažje predstavljamo. V tretjem poglavju opišemo problem matematično, dokažemo, da je NP-poln, in ga zapišemo v obliki celoštevilskega linearnega programa. V četrtem poglavju predstavimo algoritme za reševanje problema in jih primerjamo med sabo.

Math. Subj. Class. (2010): 90C10, 90C39, 90C59, 68Q17

ACM CCS (1998): F.2.2, F.1.3

Ključne besede: razporejanje potnikov, NP-polnost, celoštevilsko linearno programiranje, dinamično programiranje, sestopanje, 2-zamena, psevdokoda

Keywords: scheduling, NP-completeness, integer linear programming, dynamic programming, backtracking, 2-opt, pseudocode

1 Uvod

Turistična organizacija DISKO d.o.o. pripravlja nov program, ogled desetih najbolj priljubljenih evropskih mest (TOP 10). Posebnost programa je v tem, da skupine potnikov same določijo, kdaj in kje bodo potovanje začele ter končale, katere kraje bodo še obiskale (notranji kraji) in koliko dni bodo v vsakem od krajev, ki jih bodo obiskale. Agencija želi te notranje kraje razporediti tako, da bo zanjo končni strošek čim nižji, pri tem pa mora upoštevati določene cene in pogoje.

Ker z večanjem števila skupin in z vedno več izbranimi notranjimi kraji število možnih kombinacij eksponentno raste, bi algoritem, ki le pregleda vse možnosti in izbere najboljšo, bil časovno in prostorsko preslab, zato se bomo problema lotili z dinamičnim programiranjem in s sestopanjem. Ker pa tudi ta dva algoritma z večanjem števila skupin in večanjem števila notranjih krajev postaneta preslaba, poskusimo še, koliko se z 2-zamenami približamo optimalni rešitvi. Problem bomo poskusili zapisati tudi v obliki celoštevilskega linearnega programa.

Pokazali bomo, da je problem NP-poln (tudi, če bodisi omejimo število skupin bodisi omejimo število krajev).

Na koncu izvedemo še meritve, da vidimo, kdaj se za boljšega izkaže algoritem z dinamičnim programiranjem in kdaj se za boljšega izkaže algoritem s sestopanjem.

Diplomi je priložena zgoščenka, na kateri je celoten program (vsi algoritmi), ki je napisan v programskem jeziku C#. V mapi TOP10 je osem datotek, od tega šest s končnico .cs (vseh šest skupaj predstavlja celoten program) in dve s končnico .txt, ki predstavljata vhodne in izhodne podatke. Datoteka C.cs vsebuje spremenljivke razreda C (stran 42), njegov konstruktor in metode tega razreda. Datoteka Dinamicno.cs vsebuje algoritem z dinamičnim programiranjem (glej podrazdelek 4.3, str. 40). V datoteki Program.cs generiramo vhodne podatke (4.1, str. 27), jih zapišemo na datoteko, preberemo iz datoteke in pokličemo vse tri algoritme. Datoteka Sestopanje.cs vsebuje algoritem s sestopanjem (4.2, str. 32), datoteka SestopanjeIn2Zamena.cs vsebuje algoritem, ki najprej s sestopanjem poišče začetno dopustno rešitev, nato pa z 2-zamenami išče cenejšo rešitev (4.4, str. 47). Zadnja je datoteka Skupina.cs, ki vsebuje spremenljivke razreda Skupina (str. 32), njegov konstruktor in metode tega razreda. Datoteka vhodniPodatki.txt vsebuje vhodne podatke, datoteka izhodniPodatki.txt pa vsebuje vhodne podatke in rešitve vseh treh algoritmov ter njihove rezultate meritev. Na zgoščenki je še datoteka Diplomsko_delo_MirjamKolar.pdf, ki vsebuje diplomsko delo v elektronski obliki (format .pdf).

2 Opis problema

Kot smo povedali že v uvodu, turistična organizacija DISKO d.o.o. pripravlja nov program, ogled 10 najbolj priljubljenih evropskih mest (TOP 10). Program je namenjen potnikom s celega sveta, posebnost programa pa je v tem, da ko se prijavlja skupina potnikov, skupina sama določi, kje bo potovanje začela (eno od teh 10 mest), kdaj bo potovanje začela (kateri dan), kje bo potovanje zaključila (eno od teh 10 mest, ta kraj je lahko enak začetnemu, ni pa to pravilo), katere od ostalih krajev bo obiskala in koliko časa (dni) bo v vsakem od krajev.

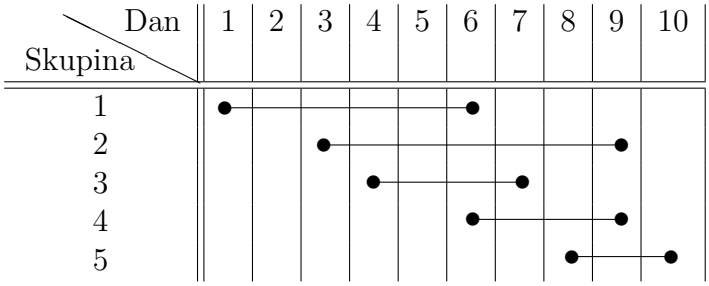
Skupina torej sama določi začetni in končni kraj, kako si povrsti sledijo ostali (izbrani) kraji za skupino, pa določi agencija sama. Agencija vsakih nekaj dni razporedi na novo prijavljene skupine, npr. enkrat tedensko ali ko se prijavi določeno število skupin, in te razporeditve se kasneje ne da spremeniti. Za vsako skupino želi agencija te ostale kraje razporediti tako, da bo njen končni strošek čim nižji, pri tem pa mora upoštevati sledeče cene in pogoje.

Pogoji
Ker potniki med kraji potujejo z letali, je pri določanju zaporedja krajev pomembno, ali obstajajo neposredne letalske povezave med kraji. Med kraji se potuje zjutraj, torej, če skupina v določen kraj prispe na določen dan, to pomeni, da skupina ta dan zjutraj leti iz prejšnjega kraja v ta kraj in ima po prihodu v kraj isti dan ogled.

Naslednji pogoj je omejitev glede števila potnikov, ki lahko na določen dan pridejo v določen kraj (npr. 50 potnikov). Namreč, ko skupina pride v določen kraj, ima tisti dan v tem kraju voden ogled skupaj z vsemi skupinami, ki pridejo tisti dan v ta kraj. Voden ogled kraja je namreč vedno prvi dan, ko skupina pride v kraj, če pa skupina ostane v kraju več dni, ima ostale dneve prosto za samostojne oglede. Omejitev imamo zaradi tega, ker imamo v kraju npr. samo enega vodiča in en avtobus. Za začetni kraj se skupina sama odloči, ali bo imela v njem voden ogled ali ne, prav tako za končni kraj. Če sta ta dva kraja enaka, se skupina sama odloči, kdaj bo imela ogled tega kraja (na začetku ali na koncu), če ga bo sploh imela. Pri tej omejitvi moramo upoštevati skupine, ki so že razporejene od prejšnjih razporejanj. Obstaja namreč možnost, da se časovni intervali potovanj že pred tem razporejenih skupin in skupin, ki jih trenutno razporejamo, prekrivajo. Omejitev glede števila postelj in prostora na letalih ni.

Cene
Prve cene so cene letalskih kart med kraji, kjer med kraji obstajajo povezave.

Druge cene pa so cene, ki jih imamo v določenem kraju na določen dan, če v ta kraj ta dan prispe vsaj ena skupina in ima v njem ogled (skupina lahko nima ogleda le v začetnem ali končnem kraju). To so cene za voden ogled. Te cene so odvisne od kraja in ne od števila potnikov, razen če je število potnikov enako 0, saj potem ogleda ni. Če pa ogled je, je vseeno, ali je potnik eden ali jih je 50, saj je cena v določenem kraju vsota stroškov za avtobus, šoferja, vodiča ipd. V primeru, da ta dan v ta kraj prispe kakšna skupina od prejšnjih razporejanj in ima v njem ogled, cene ne prištejemo, saj smo jo prišteli že pri takratnem razporejanju.



Slika 1: Primer časovnih intervalov za skupine.

Vsaka skupina bo imela torej neko začrtano pot, v nekem kraju se bo lahko združilo več skupin, ki pa bodo šle potem spet vsaka svojo pot v naslednji kraj.

Če pogledamo problem, imamo torej za vsako skupino podano število potnikov, datum odhoda, kraj odhoda (začetek), kraj prihoda (konec), množico krajev, ki jih skupina želi obiskati, in število dni bivanja v vsakem od teh krajev. Drugi podatki so še: stroški prvega dne v vsakem od krajev, cene letalskih prevozov med kraji za vsak dan, omejitev glede števila v istem dnevu prispelih potnikov v vsakem kraju in podatki o že razporejenih skupinah.

2.1 Za lažje razumevanje enostaven primer

Za začetek bomo naredili enostavnejši primer, s katerim si bomo potem lažje predstavljali, kaj je treba računati, ob njem pa bomo tudi pojasnili nekatere stvari.

Recimo, da imamo v vsakem od krajev 2400 enot/dan fiksnih stroškov ne glede na število skupin oz. število prispelih potnikov, razen če je število prispelih potnikov enako 0. Zakaj prispelih? Zato, ker imamo stroške z njimi samo prvi dan v vsakem od krajev. Cene nočitev in drugih storitev, ki spadajo zraven (vse razen cen, ki smo jih že omenili v uvodu, torej cen letalskih kart in fiksnih stroškov), so namreč na potnika neodvisne od števila potnikov v določenem kraju in od dneva, zato jih pri iskanju najcenejše rešitve ne upoštevamo oz. jih damo na 0. Poleg teh stroškov so tu še stroški za letalske karte med dvema krajema. Zato potrebujemo tabelo cen letalskih kart med kraji. Za ta primer bomo vzeli samo 4 kraje, da ne bo preveč računanja:

		London	Pariz	Rim	Berlin
London	(L)	0	20	50	40
Pariz	(P)	20	0	30	15
Rim	(R)	50	30	0	25
Berlin	(B)	40	15	25	0

Tabela 1: Cene letalskih kart med kraji.

Privzeli bomo, da nimamo omejitev glede števila postelj, saj je predvideno, da skupina potnikov ne bo ostala v določenem kraju veliko dni, ker je namen tega programa ogled čim več mest v kratkem časovnem obdobju. Na primer, če si bo skupina ogledala 5 krajev, najverjetneje ne bo v vsakem od teh krajev ostala 10 dni, pri tem imela v vsakem kraju en dan voden ogled in nato 9 dni sama pohajkovala po njem. Torej, ker je število prispelih potnikov omejeno in ker predvidevamo, da nobena skupina ne bo ostala zelo dolgo v določenem kraju, ne bo v določenem kraju naenkrat zelo veliko potnikov, torej glede števila postelj ne bo problema. Prav tako ni omejitev glede prostora na letalih. V določen kraj na določen dan leti le toliko potnikov, kot je omejitev glede števila prispelih potnikov z ogledom, in še potniki, za katere je to zadnji kraj in v njem nimajo ogleda.

Preden se lotimo reševanja primera, razložimo razliko med tem, kako bodo stroške računali algoritmi in kako bomo stroške prikazali v tem primeru. Razlika je pri načinu računanja fiksnih stroškov. V algoritmih za vsak dan za posamezen kraj, v primeru, da ta dan vanj prispe vsaj ena skupina, k skupni ceni prištejemo fiksne stroške (v tem primeru 2400 enot). V primeru pa bomo fiksne stroške v določenem kraju na določen dan razdelili na število prispelih potnikov v ta kraj ta dan in bomo potem k vsakemu potniku posebej prišteli te stroške. Tako bomo za vsako skupino dobili stroške za posameznega potnika (da dobimo predstavo, koliko približno so stroški na posameznega potnika skupine). Končno ceno bomo dobili tako, da bomo za vsako skupino stroške na potnika pomnožili s številom potnikov v skupini in sešteli te zmnožke.

Za ta primer bomo privzeli, da se nobena že prej razporejena skupina časovno ne prekriva s potovanji teh skupin. Vzeli bomo, da imamo tri skupine. Prva skupina potovanje začne in konča v Londonu ter obišče vse kraje, v njej je 20 potnikov, odhod pa imajo na dan 1. Druga skupina začne in konča v Rimu, obišče še London in Pariz, v njej je 10 potnikov, odhod pa imajo na dan 3. Tretja skupina začne in konča prav tako v Londonu, obišče še Pariz in Rim, v njej je 15 potnikov, odhod pa imajo na dan 4. Vsaka skupina je v vsakem od krajev, ki jih obišče, en dan. Pri vsaki skupini je začetni kraj enak končnemu, vsaka skupina pa ima ogled začetnega (= končnega) kraja na začetku potovanja.

Oglejmo si za vsako skupino možna zaporedja obiskov krajev in za vsako zaporedje obiskov vsoto cen letov za eno osebo.

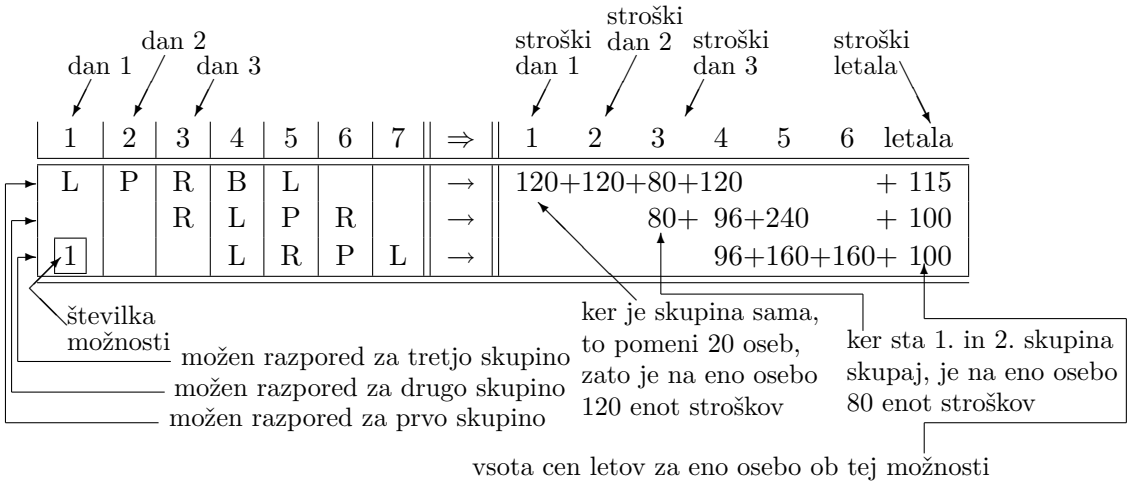
Prva skupina:
L - P - R - B - L $\rightarrow 20 + 30 + 25 + 40 = 115$ enot
L - P - B - R - L $\rightarrow 20 + 15 + 25 + 50 = 110$ enot
L - R - P - B - L $\rightarrow 50 + 30 + 15 + 40 = 135$ enot
L - R - B - P - L $\rightarrow 50 + 25 + 15 + 20 = 110$ enot
L - B - P - R - L $\rightarrow 40 + 15 + 30 + 50 = 135$ enot
L - B - R - P - L $\rightarrow 40 + 25 + 30 + 20 = 115$ enot

Druga skupina:
R - L - P - R $\rightarrow 50 + 20 + 30 = 100$ enot
R - P - L - R $\rightarrow 30 + 20 + 50 = 100$ enot

Tretja skupina
L - P - R - L $\rightarrow 20 + 30 + 50 = 100$ enot
L - R - P - L $\rightarrow 50 + 30 + 20 = 100$ enot

Za lažji pregled bomo naredili tabelo vseh možnosti. Kot smo videli malo prej, je za prvo skupino 6 možnosti, za drugo skupino sta 2 možnosti, prav tako sta 2 možnosti za tretjo skupino, torej imamo vsega skupaj: $6 \cdot 2 \cdot 2 = 24$ možnosti.

Najprej pa na primeru pojasnimo, kaj pomenijo vrednosti v tabeli 2:



1	2	3	4	5	6	7	⇒	1	2	3	4	5	6	letala
L	P	R	B	L			→	120+120+80+120						+ 115
		R	L	P	R		→			80+	96+	96		+ 100
2			L	P	R	L	→				96+	96+	160+	100

ker sta 2. in 3. skupina skupaj, je za njiju na eno osebo 96 enot stroškov, 1. skupina pa je sama, zato je pri tej skupini na eno osebo 120 enot stroškov

sicer je res, da sta 2. in 3. skupina v istem kraju, a za 2. skupino je to konec potovanja in nima ogleda v tem kraju, saj ga je imela že prvi dan. Zato ima ogled le 3. skupina, torej 15 ljudi in pride na eno osebo 160 enot stroškov

Slika 2: Razlaga pomena vrednosti v tabeli 2.

1	2	3	4	5	6	7	⇒	1	2	3	4	5	6	letala
L	P	R	B	L			→	120+	120+	80+	120			+115
		R	L	P	R		→			80+	96	+240		+100
1			L	R	P	L	→				96	+160	+160	+100
L	P	R	B	L			→	120+	120+	80+	120			+115
		R	L	P	R		→			80+	96	+ 96		+100
2			L	P	R	L	→				96	+ 96	+160	+100
L	P	R	B	L			→	120+	120+	80+	120			+115
		R	P	L	R		→			80	+240	+240		+100
3			L	R	P	L	→				160	+160	+160	+100
L	P	R	B	L			→	120+	120+	80+	120			+115
		R	P	L	R		→			80	+240	+240		+100
4			L	P	R	L	→				160	+160	+160	+100
L	P	B	R	L			→	120+	120+	120+	120			+110
		R	L	P	R		→			240+	96	+240		+100
5			L	R	P	L	→				96	+160	+160	+100
L	P	B	R	L			→	120+	120+	120+	120			+110
		R	L	P	R		→			240+	96	+ 96		+100
6			L	P	R	L	→				96	+ 96	+160	+100
L	P	B	R	L			→	120+	120+	120+	120			+110
		R	P	L	R		→			240	+240	+240		+100
7			L	R	P	L	→				160	+160	+160	+100
L	P	B	R	L			→	120+	120+	120+	120			+110
		R	P	L	R		→			240	+240	+240		+100
8			L	P	R	L	→				160	+160	+160	+100
L	R	P	B	L			→	120+	120+	120+	120			+135
		R	L	P	R		→			240+	96	+240		+100
9			L	R	P	L	→				96	+160	+160	+100
L	R	P	B	L			→	120+	120+	120+	120			+135
		R	L	P	R		→			240+	96	+ 96		+100
10			L	P	R	L	→				96	+ 96	+160	+100
L	R	P	B	L			→	120+	120+	120+	120			+135
		R	P	L	R		→			240	+240	+240		+100
11			L	R	P	L	→				160	+160	+160	+100
L	R	P	B	L			→	120+	120+	120+	120			+135
		R	P	L	R		→			240	+240	+240		+100
12			L	P	R	L	→				160	+160	+160	+100
L	R	B	P	L			→	120+	120+	120+	120			+110
		R	L	P	R		→			240+	96	+240		+100
13			L	R	P	L	→				96	+160	+160	+100
L	R	B	P	L			→	120+	120+	120+	120			+110
		R	L	P	R		→			240+	96	+ 96		+100
14			L	P	R	L	→				96	+ 96	+160	+100
L	R	B	P	L			→	120+	120+	120+	80			+110
		R	P	L	R		→			240+	80	+240		+100
15			L	R	P	L	→				160	+160	+160	+100

1	2	3	4	5	6	7	⇒	1	2	3	4	5	6	letala
L	R	B	P	L			→	120+	120+	120+	80			+110
		R	P	L	R		→				240+	80+	240	+100
16			L	P	R	L	→					160+	160+	160+100
L	B	P	R	L			→	120+	120+	120+	120			+135
		R	L	P	R		→				240+	96+	240	+100
17			L	R	P	L	→					96+	160+	160+100
L	B	P	R	L			→	120+	120+	120+	120			+135
		R	L	P	R		→				240+	96+	96	+100
18			L	P	R	L	→					96+	96+	160+100
L	B	P	R	L			→	120+	120+	120+	120			+135
		R	P	L	R		→				240+	240+	240	+100
19			L	R	P	L	→					160+	160+	160+100
L	B	P	R	L			→	120+	120+	120+	120			+135
		R	P	L	R		→				240+	240+	240	+100
20			L	P	R	L	→					160+	160+	160+100
L	B	R	P	L			→	120+	120+	80+	120			+115
		R	L	P	R		→				80+	96+	240	+100
21			L	R	P	L	→					96+	160+	160+100
L	B	R	P	L			→	120+	120+	80+	120			+115
		R	L	P	R		→				80+	96+	96	+100
22			L	P	R	L	→					96+	96+	160+100
L	B	R	P	L			→	120+	120+	80+	80			+115
		R	P	L	R		→				80+	80+	240	+100
23			L	R	P	L	→					160+	160+	160+100
L	B	R	P	L			→	120+	120+	80+	80			+115
		R	P	L	R		→				80+	80+	240	+100
24			L	P	R	L	→					160+	160+	160+100

Tabela 2: Tabela vseh možnosti.

V spodnjo tabelo bomo za vsako od teh 24 možnosti napisali, kolikšna je vsota za posamezno osebo, kolikšna je vsota glede na število oseb in kolikšna po vrsti je ta druga vsota glede na višino cene.

Vrednosti v tabeli pomenijo:

	1. sk.	2. sk.	3. sk.	vsota glede na število potnikov	
1	555	516	516	$555 \cdot 20 + 516 \cdot 10 + 516 \cdot 15 = 24000$	5

kolikšna po vrsti je vsota glede na število potnikov glede na višino cene

$120 + 120 + 80 + 120 + 115 = 555$ enot so celotni stroški za eno osebo iz prve skupine

Slika 3: Kaj pomenijo vrednosti v tabeli 3.

	1. sk.	2. sk.	3. sk.	vsota glede na število potnikov	
1	555	516	516	$555 \cdot 20 + 516 \cdot 10 + 516 \cdot 15 = 24000$	5
2	555	372	452	$555 \cdot 20 + 372 \cdot 10 + 452 \cdot 15 = 21600$	**1**
3	555	660	580	$555 \cdot 20 + 660 \cdot 10 + 580 \cdot 15 = 26400$	15
4	555	660	580	$555 \cdot 20 + 660 \cdot 10 + 580 \cdot 15 = 26400$	15
5	590	676	516	$590 \cdot 20 + 676 \cdot 10 + 516 \cdot 15 = 26300$	11
6	590	532	452	$590 \cdot 20 + 532 \cdot 10 + 452 \cdot 15 = 23900$	3
7	590	820	580	$590 \cdot 20 + 820 \cdot 10 + 580 \cdot 15 = 28700$	19
8	590	820	580	$590 \cdot 20 + 820 \cdot 10 + 580 \cdot 15 = 28700$	19
9	615	676	516	$615 \cdot 20 + 676 \cdot 10 + 516 \cdot 15 = 26800$	17
10	615	532	452	$615 \cdot 20 + 532 \cdot 10 + 452 \cdot 15 = 24400$	9
11	615	820	580	$615 \cdot 20 + 820 \cdot 10 + 580 \cdot 15 = 29200$	**21**
12	615	820	580	$615 \cdot 20 + 820 \cdot 10 + 580 \cdot 15 = 29200$	**21**
13	590	676	516	$590 \cdot 20 + 676 \cdot 10 + 516 \cdot 15 = 26300$	11
14	590	532	452	$590 \cdot 20 + 532 \cdot 10 + 452 \cdot 15 = 23900$	3
15	550	660	580	$550 \cdot 20 + 660 \cdot 10 + 580 \cdot 15 = 26300$	11
16	550	660	580	$550 \cdot 20 + 660 \cdot 10 + 580 \cdot 15 = 26300$	11
17	615	676	516	$615 \cdot 20 + 676 \cdot 10 + 516 \cdot 15 = 26800$	17
18	615	532	452	$615 \cdot 20 + 532 \cdot 10 + 452 \cdot 15 = 24400$	9
19	615	820	580	$615 \cdot 20 + 820 \cdot 10 + 580 \cdot 15 = 29200$	**21**
20	615	820	580	$615 \cdot 20 + 820 \cdot 10 + 580 \cdot 15 = 29200$	**21**
21	555	516	516	$555 \cdot 20 + 516 \cdot 10 + 516 \cdot 15 = 24000$	5
22	555	372	452	$555 \cdot 20 + 372 \cdot 10 + 452 \cdot 15 = 21600$	**1**
23	515	500	580	$515 \cdot 20 + 500 \cdot 10 + 580 \cdot 15 = 24000$	5
24	515	500	580	$515 \cdot 20 + 500 \cdot 10 + 580 \cdot 15 = 24000$	5

Tabela 3: Vsota za posamezno osebo, vsota glede na število potnikov in katera po vrsti je vsota glede na višino cene.

V tabeli smo poudarili najboljši možnosti (2. in 22. možnost) ter najslabše možnosti (11., 12., 19. in 20. možnost). Zanimata nas seveda najboljši možnosti, torej 2. in 22. možnost, zato si ju še enkrat poglejmo:

1	2	3	4	5	6	7	$\Rightarrow$	1	2	3	4	5	6	letala
L	P	R	B	L			$\rightarrow$	120+	120+	80+	120			+115
		R	L	P	R		$\rightarrow$			80+	96+	96		+100
2			L	P	R	L	$\rightarrow$				96+	96+	160+	100	
L	B	R	P	L			$\rightarrow$	120+	120+	80+	120			+115
		R	L	P	R		$\rightarrow$			80+	96+	96		+100
22			L	P	R	L	$\rightarrow$				96+	96+	160+	100	

Tabela 4: Najboljši možnosti.

Vidimo torej, da dobimo najboljšo skupno vsoto pri vsoti cen letalskih kart $115 + 100 + 100 = 315$ enot, kar pa ni najcenejša vsota cen letalskih kart. Torej res ne bo dovolj gledati samo cen letalskih kart, ampak bo očitno pomembno tudi, kako bodo razporejene skupine, torej da bo čim več potnikov na isti dan prišlo v določen kraj.

2.2 Če bi pregledali vse možnosti

Poglejmo si, kako bi bilo, če bi iskali najcenejšo rešitev tako, da bi napisali program, ki bi pregledal vse možnosti. Najprej pa izračunajmo, koliko je dejansko vseh možnosti. Podatki, ki jih potrebujemo za število vseh možnosti, so število skupin in za vsako skupino število krajev, v katere ta skupina gre.

Imamo m skupin, n_i pa je število krajev, v katere gre skupina $i, i = 1, \ldots, m$. Pri vsaki skupini sta prvi in zadnji kraj določena in če vzamemo slabšo od možnosti za število možnosti, torej da sta začetni in končni kraj skupine enaka, potem imamo za skupino i $(n_i - 1)!$ število možnosti, kako bo skupina i povrsti obiskovala mesta. Če imamo m skupin, je torej vseh možnosti:

$$\prod_{i=1}^{m}(n_i - 1)!$$

Če pogledamo primer iz prejšnjega razdelka, imamo 3 skupine, prva skupina obišče štiri kraje, druga skupina obišče tri kraje, tretja skupina prav tako obišče tri kraje. Podatki so torej: $m = 3$, $n_1 = 4$, $n_2 = 3$, $n_3 = 3$.

Dobimo število možnosti:

$$\prod_{i=1}^{m}(n_i - 1)! = \prod_{i=1}^{3}(n_i - 1)! = (4-1)! \cdot (3-1)! \cdot (3-1)! = 6 \cdot 2 \cdot 2 = 24$$

S tabelo poglejmo nekaj kombinacij, da vidimo, koliko je možnosti glede na število krajev in glede na število oseb (predpostavimo, da gre vsaka skupina v vse kraje):

št. krajev	1 skupina	2 skupini	3 skupine
1	$0! = 1$	$1 \cdot 1 = 1$	$1 \cdot 1 \cdot 1 = 1$
2	$1! = 1$	$1 \cdot 1 = 1$	$1 \cdot 1 \cdot 1 = 1$
4	$3! = 6$	$6 \cdot 6 = 36$	$6 \cdot 6 \cdot 6 = 216$
6	$5! = 120$	$120^2 = 1{,}44 \cdot 10^4$	$120^3 = 1{,}728 \cdot 10^6$
8	$7! = 5040$	$5040^2 \doteq 2{,}54 \cdot 10^7$	$5040^3 \doteq 1{,}28 \cdot 10^{11}$
10	$9! = 362880$	$362880^2 \doteq 1{,}317 \cdot 10^{11}$	$362880^3 \doteq 4{,}778 \cdot 10^{16}$

št. krajev	10 skupin	100 skupin
1	$1^{10} = 1$	$1^{100} = 1$
2	$1^{10} = 1$	$1^{100} = 1$
4	$6^{10} = 60466176$	$6^{100} \doteq 6{,}533186 \cdot 10^{77}$
6	$120^{10} \doteq 6{,}1917 \cdot 10^{20}$	$120^{100} \doteq 8{,}281797 \cdot 10^{207}$
8	$5040^{10} \doteq 1{,}05756 \cdot 10^{37}$	$5040^{100} \doteq 1{,}75 \cdot 10^{370}$
10	$362880^{10} \doteq 3{,}9594 \cdot 10^{55}$	$362880^{100} \doteq 9{,}46898 \cdot 10^{555}$

Tabela 5: Število kombinacij glede na število skupin in glede na število krajev.

Vidimo, da število možnosti z večanjem števila izbranih krajev in z večanjem števila skupin eksponentno narašča. Zato pisanje algoritma, ki pregleda vse možnosti in poišče najcenejšo, niti ni smiselno, saj bi že pri malo skupinah in majhnem številu krajev bilo vseh kombinacij preveč, in bi bil algoritem časovno in prostorsko preslab.

3 Opis problema matematično

3.1 Matematični model

Sedaj, ko smo problem razložili na primeru, ga opišimo še matematično.

Osnovni vhodni podatki so:

$\mathbf{N}$	množica vseh krajev
n	število vseh krajev, $n \in \mathbb{N}$, $n = \lvert\mathbf{N}\rvert$ (n je majhen, npr. $n \leq 10$)
m	število skupin potnikov, ki jih želimo razporediti, $m \in \mathbb{N}$ ($m \leq 50$)

Če i označuje skupino i, $i = 1, \ldots, m$, velja:

$\mathbf{N}_i$	množica krajev, ki jih želi obiskati skupina i, $\mathbf{N}_i \subseteq \mathbf{N}$
n_i	število krajev, ki jih želi obiskati skupina i, $n_i = \lvert\mathbf{N}_i\rvert$
Z_i	začetni kraj za skupino i, $Z_i \in \mathbf{N}_i$
K_i	končni kraj za skupino i, $K_i \in \mathbf{N}_i$
z_i	čas (dan) začetka potovanja za skupino i
q_i	število potnikov v skupini i
$t_i : \mathbf{N}_i \to \mathbb{N}_0$	funkcija, ki za skupino i pove, koliko časa bo skupina v posameznem kraju

Za matematični opis bomo predpostavili, da sta začetni in končni kraj skupine različna ($Z_i \neq K_i$ za $i = 1, \ldots, m$). V praksi lahko upoštevamo možnost, da je začetni kraj enak končnemu. V tem primeru kraj podvojimo, skupina pa se odloči, kdaj bo imela ogled v njem (na začetku ali na koncu), če ga bo sploh imela. Če je $t_i(Z_i) = 0$, to pomeni, da skupina i v začetnem kraju nima ogleda. Skupina torej že na dan z_i leti v prvi notranji kraj in ima ta dan v njem ogled. Če je $t_i(K_i) = 0$, skupina v končnem kraju nima ogleda. V tem primeru skupina le še prileti v končni kraj in s pristankom zaključi potovanje. V tem primeru se dan, ko skupina prileti v končni kraj, ne šteje več v časovni interval potovanja. Predpostavili bomo, da v notranjih krajih skupine morajo ostati vsaj en dan, torej $t_i(A) \neq 0$ za $A \in \mathbf{N}_i \backslash \{Z_i, K_i\}$. Če je skupina v določenem kraju vsaj en dan, ima v njem zagotovo ogled.

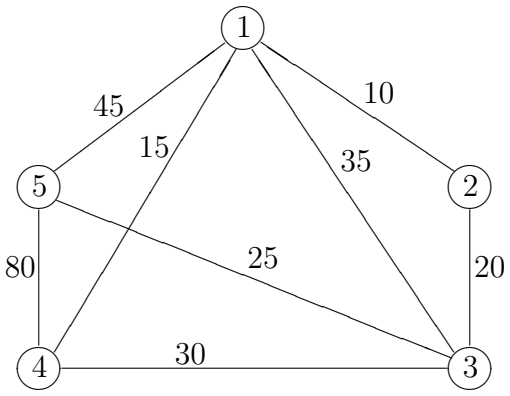
Dodatni podatki, ki nam preprečujejo, da bi skupine poljubno razporedili, so:

$p : \mathbf{N} \to \mathbb{N}$	$p(A)$ je zgornja meja za število potnikov, ki lahko na isti dan pridejo v kraj $A \in \mathbf{N}$ in imajo v njem ogled
$q : \text{dnevi} \times \mathbf{N} \to \mathbb{N}_0$	$q(d, A)$ predstavlja število potnikov, ki še lahko prispejo na dan d v kraj A in imajo v njem ogled (ustreza temu, da od $p(A)$ odštejemo število potnikov, ki imajo na dan d v kraju A ogled in so to potniki že prej razporejenih skupin)
c	dnevni stroški, ki jih imamo v posameznem kraju, če ta dan v ta kraj pride vsaj ena skupina in ima v njem ogled in nima ta dan v tem kraju ogleda že kakšna skupina iz prejšnjih razporejanj; sicer stroškov ni

Ko razporejamo skupine, moramo biti torej pozorni, da število prispelih potnikov, ki imajo ogled, v kraju A na dan d, ni večje od $q(d, A)$. Ko preverjamo, ali na dan d za kraj A k skupni ceni prištejemo c, primerjamo $q(d, A)$ in $p(A)$. Če $q(d, A) = p(A)$, c prištejemo, če pa $q(d, A) \neq p(A)$, cene c ne prištejemo.

Vhodni podatki so še cene letalskih povezav med kraji. Možne lete in cene med kraji predstavimo z uteženim grafom ali matriko $\mathcal{C}$:

graf:



Slika 4: Graf letov.

grafu prirejena matrika $\mathcal{C}$:

$$\begin{pmatrix} 0 & 10 & 35 & 15 & 45 \\ 10 & 0 & 20 & \infty & \infty \\ 35 & 20 & 0 & 30 & 25 \\ 15 & \infty & 30 & 0 & 80 \\ 45 & \infty & 25 & 80 & 0 \end{pmatrix}$$

Slika 5: Grafu letov prirejena matrika (vrednost ∞ pomeni, da ni leta).

Graf smo predstavili kot neusmerjen, vendar je graf lahko tudi usmerjen in celo časovno odvisen (cena povezave med dvema krajema je odvisna od dneva, ko se potuje med krajema). Prav tako bi se lahko tudi dnevni stroški c razlikovali od kraja do kraja in bi bili celo časovno odvisni.

Pogoj je, da želimo direktne lete, torej se točke ne smejo ponavljati.

Iz vhodnih podatkov lahko izračunamo čas zaključka potovanja k_i za skupino i:

$$k_i = z_i - 1 + \sum_{A \in \mathbf{N}_i} t_i(A).$$

Dopustna rešitev je množica funkcij $g_1, \ldots, g_m$, kjer je:

$$g_i : \{1, \ldots, n_i\} \to \mathbf{N}_i, \quad g_i(1) = Z_i, \quad g_i(n_i) = K_i.$$

Funkcija g_i mora biti bijektivna, pove pa nam, kako bo skupina i povrsti obiskovala kraje. Torej, če je $g_3(4) = A$, to pomeni, da bo četrti obiskani kraj skupine 3 kraj A. Pri ostalih pogojih za dopustnost in pri kriterijski funkciji bomo uporabili naslednji pomožni oznaki:

$$obisk(i, d, A) = \begin{cases} 1 & ; \text{ če } A \in \mathbf{N}_i \text{ in } z_i + \sum_{l=1}^{j-1} t(g_i(l)) = d \text{ in } t_i(A) \neq 0, \\ & \quad \text{kjer } j = g_i^{-1}(A), \\ 0 & ; \text{ sicer,} \end{cases}$$

$$pogoj(d, A) = \begin{cases} 1 & ; \text{ če } q(d, A) = p(A) \text{ in } \sum_{i=1}^{m} obisk(i, d, A) \geq 1, \\ 0 & ; \text{ sicer.} \end{cases}$$

Vrednost $obisk(i, d, A)$ nam torej pove, ali ima skupina i v kraju A na dan d ogled, vrednost $pogoj(d, A)$ pa, ali ima na dan d v kraju A ogled vsaj ena skupina in nima ta dan v tem kraju ogleda kakšna od že prej razporejenih skupin.

Poleg pogoja, da mora biti funkcija g_i bijektivna, mora za dopustnost veljati tudi:

$$\sum_{i=1}^{m} (obisk(i, d, A) \cdot q_i) \leq q(d, A), \quad \forall A \in \mathbf{N}, \quad d = \min_i z_i, \ldots, \max_i k_i.$$

Kriterijska funkcija f je vsota celotnih stroškov za vse skupine, iščemo pa minimum:

$$f(g_1, \ldots, g_m) = \sum_{i=1}^{m} (\sum_{j=1}^{n_i - 1} \mathcal{C}_{g_i(j), g_i(j+1)}) \cdot q_i + \sum_{d=\min_i z_i}^{\max_i k_i} \sum_{A \in \mathbf{N}} pogoj(d, A) \cdot c; \quad \text{iščemo min } f.$$

Kriterijska funkcija lahko računa tudi z neskončno. Namreč, če med krajema A_1 in A_2 ne obstaja letalska povezava, je $\mathcal{C}_{A_1, A_2} = \infty$. Če je vrednost kriterijske funkcije enaka neskončno, to pomeni, da med dvema krajema ne obstaja letalska povezava, torej rešitev ni dopustna. Če je min $f = \infty$, to pomeni, da naloga nima dopustne rešitve.

3.2 Problem razporejanja potnikov je NP-poln

Če hočemo dokazati, da je problem razporejanja potnikov NP-poln, moramo najprej pokazati, da problem spada v razred NP, nato pa moramo še znan NP-poln problem polinomsko prevesti na ta problem [3]. Da dokažemo, da problem spada v razred NP, moramo pokazati, da:

1. Je odločitveni problem (odgovor je lahko le DA ali NE).
2. Za vsak vhod, ki ima odgovor DA, podamo dokaz, ki mora biti polinomske velikosti v velikosti vhoda.

3. Opišemo polinomski algoritem, ki sprejme vhodne podatke problema in dokaz ter preveri, ali je dokaz pravilen.

Označimo odločitveno varianto problema razporejanja potnikov s Π. Vhodni podatki problema Π so torej vsi vhodni podatki, ki smo jih opisali v prejšnjem razdelku, in še število ℓ. Zanima nas, ali lahko razporedimo potnike tako, da ustrezajo omejitvam in je skupna cena največ ℓ.

Trditev 3.1. $\Pi \in$ NP.

Dokaz.

1. Problem je odločitveni, saj lahko odgovorimo le z DA ali NE.

2. Kot dokaz podamo razporeditev potnikov. Ta razporeditev za vsako skupino hrani zaporedje obiskov krajev skupine.

3. V polinomskem času lahko preverimo, ali razporeditev res razporedi potnike tako, da ustrezajo omejitvam in je skupna cena največ ℓ. Ta algoritem je polinomski, saj moramo le preveriti, ali obstajajo neposredne povezave med kraji v takem zaporedju, kot ga da rešitev, in če število potnikov v vsakem kraju vsak dan ustreza omejitvam, nato pa moramo poleg tega preverjanja sešteti še cene letalskih povezav in k temu prišteti še stroške ogledov ter pokazati, da je cena največ ℓ. To vse se lahko izračuna v polinomskem času.

Torej problem Π res spada v razred NP. ■

Najprej bomo poskusili pokazati, da je problem NP-poln, če je število krajev n poljubno, število skupin m pa omejimo.

Trditev 3.2. *Problem razporejanja potnikov je za omejeno število skupin* NP*-poln.*

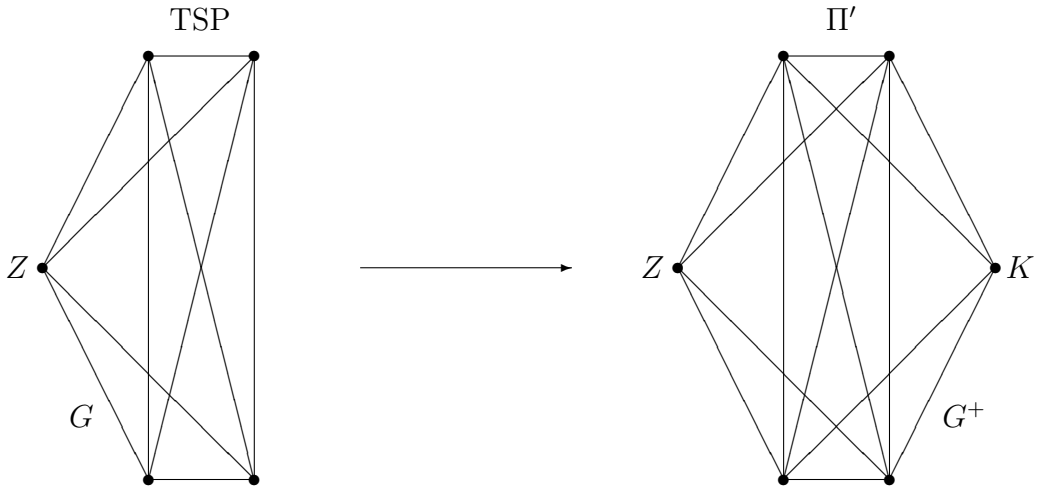
Dokaz. Da dokažemo, da je problem razporejanja potnikov za omejeno število skupin (označimo ga s Π') NP-poln, moramo znan NP-poln problem prevesti na problem Π'. Kot že znan NP-poln problem bomo vzeli problem trgovskega potnika (travelling salesman problem, TSP) [4]. Pri odločitveni varianti problema trgovskega potnika imamo podan poln graf G z množico vozlišč V, množico povezav E in dolžinami povezav $e : E(G) \longrightarrow \mathbb{N}$,

$$e_{ij} > 0 \ , \quad \forall \ (i,j) \in E,$$

ter število ω. V tem grafu je treba poiskati obhod tako, da vsako vozlišče v, $v \in V$, obiščemo natanko enkrat in je dolžina obhoda največ ω.

Za prevedbo bomo graf G, ki je poln graf z n točkami, torej $G = K_n$, prevedli na graf G^+ tako, da bomo podvojili točko Z in vse njene povezave. Dobimo torej še točko K. Stroške c v posameznem kraju bomo dali na 0, skupina naj bo le ena ($m = 1$), v njej naj bo 1 potnik, omejitev glede števila potnikov v vsakem kraju A pa je enaka številu vseh potnikov, torej 1. Cena letalske povezave med krajema i in j je enaka e_{ij}, $\ell = \omega$. Opisano prevedbo lahko izvedemo v polinomskem času.

$TSP \propto \Pi'$



$(G) \longmapsto (G^+)$, $c = 0$, $m = 1$, $q_1 = 1$, $p(A) = q_1$ za vsak kraj A, Z podvojimo

Slika 6: Prevedba problema TSP na problem Π'.

Trgovski potnik ima v grafu G obhod z dolžino največ ω natanko tedaj, ko v grafu G^+ obstaja pot od Z do K dolžine največ ℓ. ■

Pri dokazu prejšnje trditve smo predpostavili, da je število krajev n poljubno, število skupin m pa smo omejili na 1. Sedaj bi radi dokazali še nasprotno trditev. Če omejimo število krajev (recimo $n = 4$) in je število skupin poljubno, je problem še vedno NP-poln.

Trditev 3.3. *Problem razporejanja potnikov je za omejeno število krajev* NP*-poln.*

Dokaz. Za dokazovanje te trditve bomo problem Delitev (partition) prevedli na problem razporejanja potnikov za omejeno število krajev (označimo ga s Π''). Pri problemu Delitev imamo danih k naravnih števil $C_1, C_2, \ldots, C_k \in \mathbb{N}$, vprašanje pa je, ali lahko ta števila razdelimo na dva dela, ki imata enako vsoto, torej, ali obstaja podmnožica $I \subseteq \{1, 2, \ldots, k\}$, tako da je

$$\sum_{i \in I} C_i = \sum_{i \in \{1,2,\ldots,k\} \setminus I} C_i.$$

Problem Delitev je NP-poln problem [5].

Za prevedbo vzamemo poljuben vhod za Delitev: $C_1, C_2, \ldots, C_k \in \mathbb{N}$. Priredimo mu vhod za Π'' tako, da vzamemo k skupin, $m = k$, kjer ima i-ta skupina $2 \cdot C_i$ potnikov ($q_i = 2 \cdot C_i$). Vse skupine začnejo potovanje hkrati v kraju $Z \in \mathbf{N}$ in ga zaključijo v kraju $K \in \mathbf{N}$, $Z \neq K$, obiščejo še dva kraja (A_1 in A_2; $A_1, A_2 \in \mathbf{N}$), v obeh krajih so enako število dni. Stroške c v posameznem kraju damo na 0, prav tako damo na 0 cene letov, omejitev glede števila potnikov v vsakem kraju je enaka

$$p(A_1) = p(A_2) = \frac{\sum_{i=1}^{k} q_i}{2} \in \mathbb{N},$$

v začetnem in končnem kraju skupine nimajo ogleda, $\ell = 1$. To prevedbo lahko izvedemo v polinomskem času.

Trdimo, da obstaja delitev, torej obstaja $I \subseteq \{1, 2, \ldots, k\}$, da je za $C_1, \ldots, C_k \in \mathbb{N}$:

$$\sum_{i \in I} C_i = \sum_{i \in \{1,2,\ldots,k\} \setminus I} C_i,$$

natanko tedaj, ko obstaja dopustna rešitev prirejenega problema razporejanja potnikov (vsaka dopustna rešitev ima skupno ceno enako $0 < 1 = \ell$).

Najprej dokaz v desno stran. Recimo, da imamo množico I, ki reši problem delitev. Potem gredo skupine, ki pripadajo množici I, najprej v kraj A_1 in nato v kraj A_2, ostale skupine pa ravno obratno. V A_1 jih pride najprej ravno $\sum_{i \text{ je v kraju } A_1} q_i = p(A_1)$ in v A_2 druga polovica ($\sum_{i \text{ je v kraju } A_2} q_i = p(A_2)$), nato se zamenjajo. Torej, če imamo delitev in če skupine potujejo, kot smo opisali, potem je zaradi omejitev glede števila potnikov v vsakem kraju rešitev dopustna.

Za dokaz v levo stran predpostavimo, da imamo potovanje, kot smo ga opisali. V prvi kraj jih pride kvečjemu $p(A_1) = \frac{\sum_{i=1}^{k} q_i}{2}$, v drugega pa kvečjemu $p(A_2) = \frac{\sum_{i=1}^{k} q_i}{2}$. Ker morajo vse skupine že prvi dan odpotovati bodisi v A_1 bodisi v A_2, jih v oba kraja pride točno $\sum_{i=1}^{k} q_i$. To nam da delitev C-jev, kot jo želimo. ■

3.3 Zapis v obliki celoštevilskega linearnega programa

Celoštevilski linearni program [6] je matematični program, ki zadošča vsem pogojem linearnega programa, le da so vse spremenljivke omejene na podmnožice celih števil. Torej, o celoštevilskem linearnem programu govorimo, ko gre za nalogo:

Poišči minimum (ali maksimum) funkcije $c^T x$ pri pogojih: $Ax \leqslant b$, $x \geqslant 0$, kjer je x celoštevilski vektor ustrezne velikosti, c in b sta realna vektorja ustreznih velikosti in A je realna matrika ustrezne velikosti.

Po tem, ko smo v matematičnem opisu podali dopustno rešitev, bi pričakovali, da bodo spremenljivke celoštevilskega linearnega programa funkcije g_i iz dopustne rešitve. Vendar bomo namesto njih uporabili 0/1 spremenljivke. Torej imamo 0/1 linearni program.

Pri opisu 0/1 linearnega programa bomo uporabili naslednje oznake:

Skupine bomo označevali z indeksom i, kjer $i \in \{1, \ldots, m\}$.

Kraje bomo označili s števili od 1 do n, tekli bodo po indeksu j, torej $j \in \{1, \ldots, n\}$.

Z Δ bomo označili en dan po tem, ko skupina, ki zadnja zaključi potovanje, zadnjič prespi. Namreč, če ta skupina le leti v končni kraj in je dolžina bivanja v njem enaka 0, potrebujemo zaradi letov še dan, ko skupina leti vanj. Pri tem predpostavimo, da $\min_i z_i = 1$. Če je v vhodnih podatkih $\min_i z_i > 1$, v predpripravi vse premaknemo tako, da je $\min_i z_i = 1$. Dan Δ torej ustreza $\max_i k_i + 1$. Dnevi bodo tekli po indeksu d, torej $d \in \{1, \ldots, \Delta\}$.

Za skupino i bomo označili množico dni, ko skupina biva v notranjih krajih, z D_i, torej $D_i = \{z_i + t_i(Z_i), \ldots, k_i - t_i(K_i)\}$.

V nadaljevanju bomo podali opis spremenljivk, povedali, čemu morajo zadoščati, da ustrezajo dopustni rešitvi, ter kdaj ima katera spremenljivka vrednost 0 in kdaj vrednost 1.

Imeli bomo osnovni vektor x, ki predstavlja dopustno rešitev oz. iz njega lahko izračunamo funkcije g_i iz dopustne rešitve. Ker pa z enačbami za x ne bomo mogli zagotoviti zaporednosti bivanja v krajih in z njim ne bomo mogli na enostaven način zapisati kriterijske funkcije, si bomo pomagali s pomožnimi spremenljivkami (vektorji y^+, y^-, z in o), ki jih bomo opisali postopno v nadaljevanju, ko jih bomo potrebovali.

Vektor x

Vektor x pove, v katerem kraju skupina biva na določen dan, torej ta dan skupina v tem kraju še prespi:

$$x_{ijd} = \begin{cases} 1 & ; \text{ če skupina } i \text{ na dan } d \text{ biva v kraju } j, \\ 0 & ; \text{ sicer.} \end{cases}$$

V x poznamo za vsako skupino za začetni in končni kraj, kdaj skupina pride vanju in dolžino bivanja v teh dveh krajih. Če skupina i na dan d biva v začetnem kraju, nastavimo $x_{iZ_id} = 1$ in $x_{ijd} = 0$ za $j \in \mathbf{N}_i \backslash \{Z_i\}$, sicer nastavimo $x_{iZ_id} = 0$. Če skupina i na dan d biva v končnem kraju, nastavimo $x_{iK_id} = 1$ in $x_{ijd} = 0$ za $j \in \mathbf{N}_i \backslash \{K_i\}$, sicer nastavimo $x_{iK_id} = 0$. Če skupina i le leti iz kraja Z_i, torej $t_i(Z_i) = 0$, je $x_{iZ_id} = 0$ za $d = 1, \ldots, \Delta$. Prav tako, če skupina i le prileti v kraj K_i, torej $t_i(K_i) = 0$, je $x_{iK_id} = 0$ za $d = 1, \ldots, \Delta$. Za x vemo tudi, katere dni skupina ni na potovanju, za tiste dneve nastavimo vrednosti za to skupino na 0. Prav tako vemo, katerih krajev skupina ne obišče, za tiste kraje damo vrednosti za to skupino v x na 0. Za nekatere spremenljivke v vektorju x torej že na začetku določimo njihove vrednosti. Za začetni in končni kraj so vse spremenljivke že določene, torej moramo določiti še vrednosti v nekaterih spremenljivkah, ki predstavljajo notranje kraje skupin. Spremenljivke morajo zadoščati pogojem, da je določena skupina lahko le v enem kraju hkrati:

$$\sum_{j \in \mathbf{N}_i \backslash \{Z_i, K_i\}} x_{ijd} = 1 \quad \text{za} \quad i = 1, \ldots, m \quad \text{in} \quad d \in D_i. \tag{3.1}$$

Veljati mora tudi, da je skupina i v kraju j točno $t_i(j)$ dni:

$$\sum_{d \in D_i} x_{ijd} = t_i(j) \quad \text{za} \quad i = 1, \ldots, m \quad \text{in} \quad j \in \mathbf{N}_i \backslash \{Z_i, K_i\}. \tag{3.2}$$

Z vsem zgoraj opisanim smo zagotovili, da je skupina le v enem kraju hkrati, da je v vsakem kraju toliko dni, kot mora biti, in da je na potovanju v pravilnem časovnem intervalu. Vendar moramo zagotoviti tudi, da je skupina i v kraju j $t_i(j)$ dni zapored. Za to si bomo pomagali z vektorjem y^+, ki predstavlja prihode v kraje, skupaj z njim pa bomo uvedli še vektor y^-, ki predstavlja odhode iz krajev.

Vektorja y^+ in y^-

Vektor y^+ označuje prihode v kraje z ogledi, torej kateri dan v določen kraj pride skupina in ima v njem ogled. Vektor y^- pa označuje dan, ko skupina odleti iz določenega kraja. To je en dan zatem, ko je skupina zadnji dan v tem kraju. Če je to začetni kraj skupine i in v njem skupina nima ogleda, je to dan z_i. Iz končnega kraja skupina ne leti:

$$y^+_{ijd} = \begin{cases} 1 & ; \text{ če skupina } i \text{ na dan } d \text{ pride v kraj } j \text{ in ima v njem ogled,} \\ 0 & ; \text{ sicer,} \end{cases}$$

$$y^-_{ijd} = \begin{cases} 1 & ; \text{ če skupina } i \text{ na dan } d \text{ odleti iz kraja } j, \\ 0 & ; \text{ sicer.} \end{cases}$$

V y^+ in y^- poznamo za vsako skupino za začetni in končni kraj, kdaj ima skupina v teh dveh krajih ogled, če ga ima, in kdaj skupina odleti iz začetnega kraja. Če ima skupina i na dan d ogled v začetnem kraju, nastavimo $y^+_{iZ_id} = 1$, sicer nastavimo $y^+_{iZ_id} = 0$. Če ima skupina i na dan d ogled v končnem kraju, nastavimo $y^+_{iK_id} = 1$, sicer nastavimo $y^+_{iK_id} = 0$. Skupina na dan $z_i + t_i(Z_i)$ odleti iz začetnega kraja, zato nastavimo $y^-_{iZ_i(z_i+t_i(Z_i))} = 1$ in za vse ostale dni d nastavimo $y^-_{iZ_id} = 0$. Ker skupina iz končnega kraja ne leti, nastavimo $y^-_{iK_id} = 0$ za $d = 1, \ldots, \Delta$. Vemo tudi, katerih krajev skupina ne obišče, za tiste kraje damo vrednosti za to skupino v y^+ in y^- na 0, in vemo, katere dni skupina i ni na potovanju, za tiste dni, razen za dan $k_i + 1$, damo vrednosti za skupino i v y^+ in y^- na 0. Torej, za nekatere spremenljivke v vektorjih y^+ in y^- že na začetku določimo njihove vrednosti. Za začetni in končni kraj so vse spremenljivke že določene, torej moramo določiti še vrednosti v nekaterih spremenljivkah, ki predstavljajo notranje kraje skupin. Vrednosti za notranje kraje v vektorjih y^+ in y^- dobimo iz vektorja x z enačbami (3.3) in (3.4):

$$-x_{ij(d-1)} + x_{ijd} - y^+_{ijd} + y^-_{ijd} = 0 \quad \text{za} \quad i = 1, \ldots, m, \quad d = z_i, \ldots, k_i + 1 \tag{3.3}$$

$$\text{in} \quad j \in \mathbf{N}_i \backslash \{Z_i, K_i\}, \text{ kjer } x_{ij(z_i-1)} = 0,$$

$$y^+_{ijd} + y^-_{ijd} \leq 1 \quad \text{za} \quad i = 1, \ldots, m, \quad d = z_i, \ldots, k_i + 1 \tag{3.4}$$

$$\text{in} \quad j \in \mathbf{N}_i \backslash \{Z_i, K_i\}.$$

Z enačbami (3.4) zagotovimo, da skupina ne more na isti dan priti v kraj, imeti v njem ogleda in odleteti iz njega.

Preverimo s tabelo, da dobimo v y^+ in y^- res pravilne vrednosti:

$x_{ij(d-1)}$	x_{ijd}	0	0	0	1	1	0	1	1
	y^+_{ijd}		0		1		0		0
	y^-_{ijd}		0		0		1		0

S pomočjo vektorja y^+ zagotovimo zaporednost bivanja v določenem kraju, torej, če je skupina i v kraju j $t_i(j)$ dni, je v tem kraju $t_i(j)$ dni zapored. Na prvi pogled se zdi, da je to zagotovljeno zaradi optimalnosti (torej najcenejše rešitve), a temu ni tako. Če si ogledamo to na preprostem primeru:

skupina i	n_i	Z_i	K_i	z_i	q_i	$t_i(1)$	$t_i(2)$	$t_i(3)$	$t_i(4)$	$t_i(5)$
1	5	1	5	1	1	1	3	1	1	1

Ostali podatki so: $c = 2400$, $p(A) = 50$ za vse $A \in \mathbf{N}$, vrednosti $q(d, A)$ pa so:

$d \backslash A$	1	2	3	4	5
1	50	50	50	50	50
2	50	**20**	50	50	50
3	50	50	**20**	50	50
4	50	**20**	50	50	50
5	50	50	**20**	**20**	50
6	50	**20**	50	50	50
7	50	50	50	50	50

Cene letov so:

$$\begin{pmatrix} 0 & 10 & 200 & 100 & 100 \\ 10 & 0 & 10 & 10 & 10 \\ 200 & 10 & 0 & 50 & 100 \\ 100 & 10 & 50 & 0 & 50 \\ 100 & 10 & 100 & 50 & 0 \end{pmatrix}$$

Račun pokaže, da je optimalna rešitev po dnevih:

dan	1	2	3	4	5	6	7
skupina 1	1	2	2	2	3	4	5

in končna cena je 7320. Če pa ne upoštevamo, da mora biti skupina v kraju j $t_1(j)$ dni zapored, dobimo cenejšo rešitev:

dan	1	2	3	4	5	6	7
skupina 1	1	2	3	2	4	2	5

.

Njena cena je 4860.

Zaradi tega moramo zagotoviti to "zaporednost" s pomočjo vektorja y^+. A kakšna naj bo enačba, da bo zagotovila, kar potrebujemo? Odgovor je preprost: skupina mora imeti toliko prihodov v notranje kraje, kolikor notranjih krajev obišče. Torej v vsak

notranji kraj pride le enkrat. "Zaporednost" bomo torej zagotovili tako, da bo število prihodov v notranje kraje enako številu notranjih krajev:

$$\sum_{d\in D_i}\sum_{j\in\mathbf{N}_i\setminus\{Z_i,K_i\}} y^+_{ijd} = n_i - 2 \quad \text{za} \quad i = 1,\ldots,m. \tag{3.5}$$

S pomočjo vektorja y^+ preverimo tudi, ali dopustna rešitev ustreza omejitvam glede števila potnikov $q(d,A)$:

$$\sum_{i=1}^{m}(y^+_{ijd}\cdot q_i) \le q(d,j) \quad \text{za} \quad j\in\mathbf{N} \quad \text{in} \quad d = 1,\ldots,\Delta. \tag{3.6}$$

Sedaj, ko smo zagotovili, da vektor x ustreza dopustni rešitvi, moramo še zapisati enačbo za kriterijsko funkcijo. Za to potrebujemo še vektorja z in o.

Vektor z

Vektor z označuje lete med kraji, torej za vsak dan določimo, ali skupina leti med dvema krajema:

$$z_{i(j_1,j_2)d} = \begin{cases} 1 & ;\ \text{če skupina } i \text{ na dan } d \text{ leti iz kraja } j_1 \text{ v kraj } j_2,\\ 0 & ;\ \text{sicer.}\end{cases}$$

Za z vemo, katere dni skupina i ni na potovanju, za tiste dneve, razen za dan k_i+1, nastavimo vrednosti za to skupino na 0. Prav tako vemo, katerih krajev skupina ne obišče, za tiste kraje pri z nastavimo za to skupino lete v te kraje in lete iz teh krajev na 0. Ker skupina ne more leteti iz določenega kraja v isti kraj, nastavimo v vektorju z vse lete iz kraja j v kraj j na 0 za vsak $j = 1,\ldots,n$. Vektor z izračunamo s pomočjo vektorjev y^+ in y^-:

$$2\cdot y^+_{ij_2d} + 2\cdot y^-_{ij_1d} - 3\cdot z_{i(j_1,j_2)d} \le 2 + b_{ij_2d} \quad \text{za} \quad i = 1,\ldots,m,\ \ d = z_i,\ldots,k_i+1 \tag{3.7}$$
$$\text{in} \quad j_1,j_2\in\mathbf{N}_i,\ \text{kjer } j_1\neq j_2,$$

$$2\cdot y^+_{ij_2d} + 2\cdot y^-_{ij_1d} - 3\cdot z_{i(j_1,j_2)d} \ge b_{ij_2d} \quad \text{za} \quad i = 1,\ldots,m,\ \ d = z_i,\ldots,k_i+1 \tag{3.8}$$
$$\text{in} \quad j_1,j_2\in\mathbf{N}_i,\ \text{kjer } j_1\neq j_2,$$

$$\text{kjer} \quad b_{ij_2d} = \begin{cases} -1 & ;\ \text{če skupina } i \text{ na dan } d \text{ prileti v kraj } j_2 \text{ in je to končni kraj}\\ & \quad \text{skupine } i,\ \text{v njem pa nima ogleda, torej } j_2 = K_i \text{ in}\\ & \quad d = z_i + \sum_{A\in\mathbf{N}_i\setminus\{K_i\}} t_i(A) \text{ in } t_i(K_i) = 0,\\ 0 & ;\ \text{sicer.}\end{cases}$$

Preverimo s tabelo, da dobimo v z res pravilne vrednosti:

$y^+_{ij_2d}$ $y^-_{ij_1d}$	0 0	0 1	1 0	1 1
$z_{i(j_1,j_2)d}$	0	0	0	1
$2\cdot y^+_{ij_2d} + 2\cdot y^-_{ij_1d} - 3\cdot z_{i(j_1,j_2)d}$	0	2	2	1
$z_{i(j_1,j_2)d},\ j_2 = K_i,\ t_i(K_i) = 0$	0	1	-	-
$2\cdot y^+_{ij_2d} + 2\cdot y^-_{ij_1d} - 3\cdot z_{i(j_1,j_2)d}$	0	−1	-	-

Vektor z uporabimo za računanje kriterijske funkcije. Ko jo računamo, ga pomnožimo s ceno leta in številom potnikov skupine, glej enačbo (3.11).

Vektor o

Vektor o pove, kdaj ima vsaj ena skupina na določen dan v določenem kraju ogled:

$$o_{jd} = \begin{cases} 1 & ; \text{ če ima na dan } d \text{ v kraju } j \text{ vsaj ena skupina ogled,} \\ 0 & ; \text{ sicer.} \end{cases}$$

Vektor o izračunamo iz y^+:

$$-\sum_{i=1}^{m} y^+_{ijd} + (m+1) \cdot o_{jd} \leq m \quad \text{za} \quad j \in \mathbf{N} \quad \text{in} \quad d = 1, \ldots, \Delta, \tag{3.9}$$

$$-\sum_{i=1}^{m} y^+_{ijd} + (m+1) \cdot o_{jd} \geq 0 \quad \text{za} \quad j \in \mathbf{N} \quad \text{in} \quad d = 1, \ldots, \Delta. \tag{3.10}$$

Preverimo s tabelo, da dobimo v o res pravilne vrednosti:

$\sum_{i=1}^{m} y^+_{ijd}$	0	m	$1, 2, \ldots, m-1$
o_{jd}	0	1	1
$-\sum_{i=1}^{m} y^+_{ijd} + (m+1) \cdot o_{jd}$	0	1	$m, m-1, \ldots, 2$

Tudi vektor o uporabimo za računanje kriterijske funkcije, pomnožimo ga z dnevnimi stroški.

Kriterijska funkcija je:

$$\sum_{i=1}^{m} \Big(\sum_{j_1 \in \mathbf{N}} \sum_{j_2 \in \mathbf{N}} \sum_{d=1}^{\Delta} z_{i(j_1,j_2)d} \cdot \mathcal{C}_{j_1,j_2} \Big) \cdot q_i + \sum_{j \in \mathbf{N}} \sum_{d=1}^{\Delta} o_{jd} \cdot c_{jd} \; ; \quad \text{iščemo min,} \tag{3.11}$$

kjer $c_{jd} = \begin{cases} 0 & ; \text{ če } q(d,j) \neq p(j), \\ c & ; \text{ sicer.} \end{cases}$

Če je vrednost kriterijske funkcije enaka neskončno, to pomeni, da rešitev ni dopustna. Obstoj letov je namreč skrit v kriterijski funkciji.

Bijekcijo zagotovimo z enačbami (3.1) - skupina je lahko le v enem kraju hkrati, (3.2) - skupina i mora biti v kraju j $t_i(j)$ dni in (3.5) - zaporednost bivanja v določenem kraju. Da ne prekoračimo omejitve glede števila prispelih potnikov z ogledom na dan d v kraj j, zagotovimo z enačbo (3.6). Kriterijsko funkcijo pa dobimo z enačbo (3.11).

Še vprašanje, kako iz teh spremenljivk dobimo funkcije g_i iz opisa dopustne rešitve. Začetni in končni kraj sta določena tudi, če v njiju ni ogleda, torej $g_i(1) = Z_i$ in $g_i(n_i) = K_i$. Ostale kraje dobimo iz x:

$$g_i(k) = j, \text{ če } x_{ijd} = 1, \text{ kjer } \; d = z_i + \sum_{l=1}^{k-1} t_i(g_i(l)) \; \text{ za } i = 1, \ldots, m \text{ in } k = 2, \ldots, n_i - 1.$$

4 Opisi algoritmov in primerjave med njimi

V tem poglavju si bomo ogledali tri različne algoritme, s katerimi lahko rešujemo problem razporejanja potnikov. Prvi algoritem je reševanje problema s sestopanjem, drugi algoritem uporablja dinamično programiranje, pri tretjem pa začetno dopustno rešitev poiščemo s sestopanjem, nato pa se z 2-zamenami skušamo čim bolj približati optimalni rešitvi.

Najprej si bomo podrobneje ogledali vsak algoritem posebej, na koncu pa bomo naredili primerjave med njimi (v kakšnih primerih se za boljšega izkaže algoritem s sestopanjem in v kakšnih primerih algoritem z dinamičnim programiranjem ter koliko se z 2-zamenami približamo optimalni rešitvi).

Na priloženi zgoščenki je celoten program (vsi algoritmi), ki je napisan v programskem jeziku C# [1]. V mapi TOP10 je osem datotek, od tega šest s končnico .cs (vseh šest skupaj predstavlja celoten program) in dve s končnico .txt, ki predstavljata vhodne in izhodne podatke. Datoteka C.cs vsebuje spremenljivke razreda C, njegov konstruktor in metode tega razreda. Datoteka Dinamicno.cs vsebuje algoritem z dinamičnim programiranjem. V datoteki Program.cs generiramo vhodne podatke, jih zapišemo na datoteko, preberemo iz datoteke in pokličemo vse tri algoritme. Datoteka Sestopanje.cs vsebuje algoritem s sestopanjem, datoteka SestopanjeIn2Zamena.cs vsebuje algoritem, ki najprej s sestopanjem poišče začetno dopustno rešitev, nato pa z 2-zamenami išče cenejšo rešitev. Zadnja je datoteka Skupina.cs, ki vsebuje spremenljivke razreda Skupina, njegov konstruktor in metode tega razreda. Datoteka vhodniPodatki.txt vsebuje vhodne podatke, datoteka izhodniPodatki.txt pa vsebuje vhodne podatke in rešitve vseh treh algoritmov ter njihove rezultate meritev.

Preden pa opišemo algoritme, opišimo postopek, s katerim lahko pripravimo testne vhodne podatke.

4.1 Generiranje vhodnih podatkov

Če želimo testirati sledeče algoritme, je potrebno napisati najprej algoritem, ki zgenerira vhodne podatke. Kot smo napisali že v razdelku 3.1, imamo za problem podane naslednje podatke:

stKrajev	...	množica vseh krajev (število krajev je enako 10), označimo jih kar od 0 do 9
konstVMestu	...	stroški, ki jih imamo v določenem kraju na določen dan, če ta dan v ta kraj pride vsaj ena skupina in ima v njem ogled
omPotnikov	...	omejitev števila prispelih potnikov z ogledom v določenem kraju na določen dan
matrikaCen	...	cene letov med kraji
stSkupin	...	število skupin

Za vse skupine imamo podane še sezname:

stPotnikov	...	število potnikov v skupinah
zacCas	...	začetki potovanj za skupine
zacKraj	...	začetni kraji za skupine
konKraj	...	končni kraji za skupine
zacDolzina	...	dolžine bivanj v začetnih krajih
konDolzina	...	dolžine bivanj v končnih krajih
ostaliKraji	...	seznami notranjih krajev, ki jih obiščejo skupine
dolzinaBivanja	...	seznami dolžin bivanj v vsakem notranjem kraju skupine

Najprej si oglejmo psevdokodo za algoritem, ki naključno zgenerira podatke in jih zapiše na datoteko:

```
nastavi konstanti za podatka:  konstVMestu, omPotnikov;

ustvari objekt za generiranje naključnih števil Random r, s pomočjo katerega
kasneje naključno določamo vhodne podatke;

z dvojno zanko for zgradi matriko cen letov med kraji;

izberi število skupin;

za vsako skupino zgeneriraj in dodaj v sezname
{
   število potnikov;
   začetek potovanja (čas);
   začetni kraj in končni kraj;

   določi dolžino bivanja v začetnem in končnem kraju (če je dolžina bivanja
   v katerem od njiju enaka 0, skupina tam nima ogleda; če je začetni kraj
   enak končnemu, ima skupina lahko ogled bodisi na začetku bodisi na koncu
   bodisi nikoli);

   določi število notranjih krajev, ki jih bo skupina obiskala = n_i;
   dokler ni izbranih n_i krajev
   { izmed krajev, ki ostanejo, izberi kraj, ki ga bo skupina obiskala; }

   za vsak kraj, ki ga skupina obišče, določi dolžino bivanja v njem;

   izračunaj konec potovanja za skupino;
}

izračunaj zadnji dan potovanja vseh skupin;
za vsak kraj za vsak dan določi omejitev glede števila potnikov;

zapiši podatke na datoteko;
```

Deli kode izgledajo tako:

```
private static void GenerirajInZapisiPodatke()
{
   const int konstVMestu = 2400;
   const int omPotnikov = 50;

   //na tem mestu inicializiramo spremenljivke, ki smo jih opisali na
   //začetku poglavja
      .
      .
      .
   List<int> konCas = new List<int>();
   int[,] KrajMeja;

   Random r = new Random(); //objekt za naključno generiranje števil

   //zgradimo cene za letalske karte med kraji
   matrikaCen = new int[stKrajev, stKrajev];
   for (int i = 0; i < stKrajev; i++)
   {
      for (int j = 0; j < stKrajev; j++)
         matrikaCen[i, j] = r.Next(20, 70);
      matrikaCen[i, i] = 0;
   }
   //za nekaj povezav določimo, da ne obstajajo
   matrikaCen = IzberiLeteKiNeObstajajo(matrikaCen);

   stSkupin = r.Next(1, 15);

   //za vsako skupino posebej zgeneriramo vse podatke
   for (int i = 0; i < stSkupin; i++)
   {
      //naredimo seznam vseh krajev, iz katerega potem izbiramo začetni,
      //končni in notranje kraje za skupino
      List<int> kraji = new List<int>();
      for (int j = 0; j < stKrajev.Count; j++) kraji.Add(j);

      stPotnikov.Add(r.Next(1, (omPotnikov / stSkupin) * 2));
      //poskrbimo, da skupine potujejo v približno enakem časovnem intervalu
      zacCas.Add(r.Next(stSkupin));
      zacKraj.Add(kraji[r.Next(stKrajev)]);
      konKraj.Add(kraji[r.Next(stKrajev)]);
      kraji.Remove(zacKraj[i]); kraji.Remove(konKraj[i]);

      //odvisno od tega, ali sta začetni in končni kraj enaka ali ne,
      //določimo, ali bo ogled na začetku in na koncu, le na začetku, le na
```

```
      //koncu ali nikoli
      int ogled = r.Next(4 - Convert.ToInt16(zacKraj[i] == konKraj[i]));
      zacDolzina.Add((ogled % 2) * r.Next(1, 5));
      konDolzina.Add(Convert.ToInt16(ogled / 2) * r.Next(1, 5));

      //dodamo notranje kraje
      int stOstalihKrajev = kraji.Count();
      List<int> ostaliKrajiZaSkupino = new List<int>();
      int steviloNotranjihKrajev = r.Next(1, stOstalihKrajev + 1);
      for (int j = 0; j < steviloNotranjihKrajev; j++)
      {
        ostaliKrajiZaSkupino.Add(kraji[r.Next(stOstalihKrajev)]);
        kraji.Remove(ostaliKrajiZaSkupino[j]);
        stOstalihKrajev--;
      }
      ostaliKraji.Add(ostaliKrajiZaSkupino);

      //določimo dolžino bivanja v notranjih krajih
      List<int> dolzinaBivanjaZaSkupino = new List<int>();
      for (int j = 0; j < ostaliKrajiZaSkupino.Count; j++)
        dolzinaBivanjaZaSkupino.Add(r.Next(1, 5));
      dolzinaBivanja.Add(dolzinaBivanjaZaSkupino);
      konCas.Add(IzracunajKonecCas(dolzinaBivanjaZaSkupino, zacCas[i],
        zacDolzina[i], konDolzina[i]));
    }

    //za vsak dan za vsak kraj določimo omejitve glede števila prispelih
    //potnikov z ogledom
    int zadnjiObiskaniDanVsehSkupin = DobiMaxInt(konCas);
    KrajMeja = new int[zadnjiObiskaniDanVsehSkupin + 1, stKrajev];
    KrajMeja = NastaviKrajMeja(KrajMeja, stKrajev, omPotnikov);

    //zapišemo podatke na vhodno datoteko
    ZapisiPodatke(stKrajev, konstVMestu, omPotnikov, matrikaCen, stSkupin,
      stPotnikov, zacCas, zacKraj, konKraj, zacDolzina, konDolzina,
      ostaliKraji, dolzinaBivanja, KrajMeja);
}
```

Razjasnimo nekatere vrstice iz kode:

`const int konstVMestu = 2400;`
Ceno bivanja v določenem kraju nastavimo na konstanto 2400.

`const int omPotnikov = 50;`
Število potnikov, ki lahko na določen dan prispe v določen kraj in ima v njem ogled, omejimo na konstanto 50.

`matrikaCen[i, j] = r.Next(20, 70);`
Cene letov med kraji naj bodo med 20 in 70 enot (`r` generira naključna števila in `r.Next(a,b)` izbere naključno celo število, večje ali enako `a` in manjše od `b`, `r.Next(a)` pa izbere naključno celo število, večje ali enako 0 in manjše od `a`).

`matrikaCen = IzberiLeteKiNeObstajajo(matrikaCen);`
Nastavi cene nekaterih letov na -1, kar pomeni, da ti leti ne obstajajo.

`List<int> kraji = new List<int>(stKrajev); for(...) ...;`
Naredi seznam krajev, vanj shrani vse kraje, v katere je možno iti. Ko izbere kraj iz tega seznama za začetni, končni ali notranji kraj, ga izbriše iz seznama. To stori s `kraji.Remove(...);`.

`stPotnikov.Add(r.Next(1, (omPotnikov / stSkupin) * 2));`
Ko za določeno skupino generiramo število potnikov, omejimo, da potnikov ni preveč, saj potem skoraj gotovo ne bi bilo dopustne rešitve.

`int ogled = r.Next(4 - Convert.ToInt16(zacKraj[i] == konKraj[i]));`
Naključno izberemo konstanto, ki pove, ali bo imela skupina ogled v začetnem in končnem kraju, le v začetnem kraju, le v končnem kraju ali v nobenem od njiju. `Convert.ToInt16(zacKraj[i] == konKraj[i])` vrne vrednost 1, če sta kraja enaka in vrne vrednost 0, če sta kraja različna. Torej, če sta kraja enaka, je lahko vrednost spremenljivke `ogled` enaka 0, 1 ali 2, če pa sta kraja različna, je lahko vrednost enaka 0, 1, 2 ali 3. Vrednost 0 pomeni, da skupina nima ogleda niti v začetnem niti v končnem kraju, vrednost 1 pomeni, da ima skupina ogled le v začetnem kraju, vrednost 2, da ima ogled le v končnem kraju, vrednost 3 pa, da ima skupina ogled v začetnem in končnem kraju.

`zacDolzina.Add((ogled % 2) * r.Next(1, 5));`
Če je vrednost spremenljivke `ogled` enaka 0 ali 2, je vrednost `ogled % 2` enaka 0, torej v začetnem kraju skupina nima ogleda in je dolžina bivanja v njem enaka 0, če pa je vrednost spremenljivke `ogled` enaka 1 ali 3, je vrednost `ogled % 2` enaka 1, torej v začetnem kraju skupina ima ogled in naključno določimo, koliko dni biva skupina v tem kraju (dolžina bivanja je med 1 in 4 dni).

`konDolzina.Add(Convert.ToInt16(ogled / 2) * r.Next(1, 5));`
Če je vrednost spremenljivke `ogled` enaka 0 ali 1, je vrednost `Convert.ToInt16(ogled / 2)` enaka 0, torej v končnem kraju skupina nima ogleda in je dolžina bivanja v njem enaka 0, če pa je vrednost spremenljivke `ogled` enaka 2 ali 3, je vrednost `Convert.ToInt16(ogled / 2)` enaka 1, torej v končnem kraju skupina ima ogled in naključno določimo, koliko dni biva skupina v tem kraju (dolžina bivanja je med 1 in 4 dni).

`ostaliKrajiZaSkupino.Add(kraji[r.Next(stOstalihKrajev)]);`
Izmed krajev, ki še ostanejo v seznamu, izberemo enega od krajev in ga damo v seznam krajev, ki jih bo skupina obiskala, nato ta kraj zbrišemo iz seznama krajev, ki so še na voljo za to skupino s `kraji.Remove(ostaliKrajiZaSkupino[j]);`.

`KrajMeja = NastaviKrajMeja(KrajMeja, stKrajev, omPotnikov);`
Za vsak kraj za vsak dan nastavimo, koliko potnikov še lahko ta dan prispe v ta kraj (ponekod damo omejitve nižje od `omPotnikov`, saj so lahko že kakšni potniki od prejšnjih razporejanj ta dan prišli v ta kraj).

4.2 Sestopanje

Sestopanje [2, 7, 8] je pristop v programiranju, kjer sistematično pregledujemo vse možnosti rešitve. Postopek sestopanja je:

- na i-tem koraku naj bo S_i množica možnih odločitev
- iščemo rešitev $x = (x_1, x_2, \ldots, x_n)$, pri čemer je $x_i \in S_i$ odločitev na i-tem koraku
- če na i-tem koraku ni več dopustnih odločitev, se vrnemo korak nazaj in izberemo drug x_{i-1}.

Sestopanje je torej sistematičen pregled drevesa vseh možnih rešitev. Rešitve problema so v listih drevesa ali pa je rešitev pot od korena do lista. Ko neka pot od korena do lista ni več obetavna, se vrnemo nazaj in poskusimo po drugi poti.

Da bomo lahko v algoritmih sploh kaj računali, potrebujemo razred Skupina, v katerem imamo za določeno skupino shranjene vse podatke (število potnikov skupine, začetek potovanja za skupino, katere kraje želi skupina obiskati in dolžino bivanja v teh krajih ter katera povrsti je ta skupina, ko beremo podatke iz datoteke (da vemo na koncu za izpis rešitve)).

Kako dela algoritem s sestopanjem:

V splošnem pri sestopanju rešitev problema iščemo korakoma. Pri sestavljanju algoritma si pomagamo z algoritmom za razporejanje kraljic na šahovsko desko (glej [2]). Za vsako skupino i posebej iščemo vektor x_i velikosti $|x_i| = n_i$. Iščemo povrsti po skupinah.

Najprej dodamo vse začetne in končne kraje (če lahko), saj so ti ves čas fiksni (dan, kraj), in tam, kjer skupine imajo v katerem od teh krajev ogled, od $q(d_{iA}, A), A \in \{Z_i, K_i\}$, odštejemo q_i. Če začetnih in končnih krajev ne moremo dodati, končamo (ni rešitve).

Torej, če imamo m skupin, in zanje vektorje $x_1, x_2, \ldots, x_m$, smo za vsak vektor določili prvo in zadnjo komponento. Označimo število notranjih krajev skupine i z j_i.

Nato začnemo pri prvi skupini. Recimo, da skupina želi obiskati notranje kraje $n_{11}, n_{12}, \ldots, n_{1j_1}$. Izbiramo kraj za x_{1_2}. Izračunamo, na kateri dan pride skupina v ta kraj $(z_1 + t_1(Z_1))$. Poskusimo, ali je lahko $x_{1_2} = n_{11}$. Če ta dan v ta kraj lahko pride še q_1 potnikov in če lahko letimo iz začetnega kraja v ta kraj, nastavimo $x_{1_2} = n_{11}$. Sicer poskusimo enako za n_{12} itd., dokler ne pridemo do kraja n_{1k}, ki ustreza tem pogojem in nastavimo $x_{1_2} = n_{1k}$.

Nato izbiramo kraj za x_{1_3}. Ponovno izračunamo, na kateri dan pride skupina v ta kraj $(z_1 + t_1(Z_1) + t_1(x_{1_2}))$. Poskusimo $x_{1_3} = n_{11}$. Spet, če ta dan v ta kraj lahko pride še q_1 potnikov in če lahko letimo iz kraja x_{1_2} v ta kraj in če ta kraj ni že izbran, torej $x_{1_2} \neq n_{11}$, nastavimo $x_{1_3} = n_{11}$. Sicer poskusimo enako za n_{12} (tu mora veljati $x_{1_2} \neq n_{12}$) itd., dokler ne pridemo do kraja n_{1k}, ki ustreza tem pogojem in nastavimo $x_{1_3} = n_{1k}$.

Torej, ko izbiramo kraj za x_{1_r}, izračunamo, kateri dan pride skupina v ta kraj $(z_1 + t_1(Z_1) + \sum_{i=2}^{r-1} t_1(x_{1_i}))$. Začnemo z n_{11}. Če ta dan v ta kraj lahko pride še q_1 potnikov in če lahko letimo iz kraja $x_{1_{r-1}}$ v ta kraj in če ne velja $(x_{1_2} = n_{11} \mid\mid x_{1_3} = n_{11} \mid\mid \ldots \mid\mid x_{1_{r-1}} = n_{11})$, nastavimo $x_{1_r} = n_{11}$. Sicer nadaljujemo dalje za n_{12} (ne velja $(x_{1_2} = n_{12} \mid\mid x_{1_3} = n_{12} \mid\mid \ldots \mid\mid x_{1_{r-1}} = n_{12})$) itd., dokler ne pridemo do kraja n_{1k}, ki ustreza tem pogojem in nastavimo $x_{1_r} = n_{1k}$.

Ko izbiramo zadnji notranji kraj, moramo preverjati še, ali lahko letimo iz tega kraja v končni kraj.

Ko pridemo do konca prve skupine, znižamo omejitve za vse notranje kraje za ustrezne dneve. Nato nadaljujemo enako za vsako naslednjo skupino.

1. Če najdemo rešitev za vse skupine, izračunamo ceno te rešitve, in če je ta rešitev cenejša od trenutno najboljše, nastavimo to rešitev za najboljšo.

Nato iščemo naslednjo rešitev tako: recimo, da je za zadnjo skupino vektor zadnje rešitve enak $(Z_m, x_{m_1}, x_{m_2}, \ldots, x_{m_{j_m}}, K_m)$ in $x_{m_{j_m}} = n_{mk}$. Potem zvišamo omejitve za to skupino za vse notranje kraje za ustrezne dneve, zbrišemo kraj n_{mk} iz tega vektorja in poskusimo nastaviti $x_{m_{j_m}} = n_{m(k+1)}$ tako kot prej. In tako nadaljujemo ...

2. Če pri nekem x_{i_r} ne moremo nastaviti nobenega kraja, gremo na $x_{i_{r-1}} = n_{ik}$, ta kraj odstranimo in poskusimo nastaviti $x_{i_{r-1}} = n_{i(k+1)}$ tako kot prej. In tako nadaljujemo ...

Če pri skupini i ne moremo pri x_{i_1} nastaviti nobenega kraja, gremo eno skupino nazaj, zvišamo omejitve za to skupino za vse notranje kraje za ustrezne dneve in

iščemo za $x_{(i-1)_{j_{(i-1)}}}$ naslednji ustrezni kraj tako kot prej. In tako nadaljujemo ...

Če pridemo do prve skupine in ne moremo nastaviti x_{1_1}, končamo.

Najboljša rešitev je shranjena (ali pa je sploh ni).

Pa si poglejmo na primeru, kako dela algoritem s sestopanjem.

4.2.1 Primer

Podanih imamo 5 krajev in 3 skupine, torej $n = 5$ in $m = 3$ (kraje in skupine bomo pri algoritmih označevali od 0 dalje in ne od 1, kot pri matematičnem opisu, saj se v računalništvu vse začne z indeksom 0; zato bomo tudi v primerih algoritmov označevali kraje in skupine od 0 dalje). Podatki so:

skupina i	n_i	Z_i	K_i	z_i	q_i	$t_i(0)$	$t_i(1)$	$t_i(2)$	$t_i(3)$	$t_i(4)$
0	5	0	4	2	25	1	1	1	1	0
1	4	2	0	3	10	1	1	1	2	-
2	4	1	2	6	11	1	3	1	4	-

Ostali podatki so: $c = 2400$, $p(A) = 50$ za vse $A \in \mathbf{N}$, vrednosti $q(d, A)$ pa so:

$d \backslash A$	0	1	2	3	4
0	50	50	50	50	50
1	**30**	50	50	50	50
2	50	**20**	50	50	50
3	50	50	**20**	50	50
4	**25**	50	50	**30**	50
5	50	**25**	50	50	50
6	50	50	**35**	50	50
7	50	50	50	50	50
$\vdots$	50	50	50	50	50
14	50	50	50	50	50

Cene letov so:

$$\begin{pmatrix} 0 & 20 & \infty & 40 & 10 \\ 20 & 0 & 30 & 15 & 20 \\ \infty & 30 & 0 & 25 & \infty \\ 40 & 15 & 25 & 0 & 40 \\ 10 & 20 & \infty & 40 & 0 \end{pmatrix}$$

Najprej poskusimo dodati za vse skupine začetni in končni kraj:

izračunamo prihode v končni kraj:

skupina i	k_i
0	6
1	7
2	14

Torej moramo pogledati, ali lahko v kraje A na dneve d pride še q_i potnikov:

A	d	q_i
0	2	25
2	3	10
1	6	11
4	6	0 (ni ogleda)
0	7	10
2	14	11

Ugotovimo, da lahko dodamo začetne in končne kraje, in spremenimo $q(d, A)$:

$d \backslash A$	0	1	2	3	4
0	50	50	50	50	50
1	30	50	50	50	50
2	**25**	20	50	50	50
3	50	50	**10**	50	50
4	25	50	50	30	50
5	50	25	50	50	50
6	50	**39**	35	50	50
7	**40**	50	50	50	50
8	50	50	50	50	50
⋮	50	50	50	50	50
13	50	50	50	50	50
14	50	50	**39**	50	50

Sedaj začnemo s skupino 0: zanjo imamo vektor x_0, $|x_0| = 5$, $x_{0_0} = 0$ in $x_{0_4} = 4$. Določiti moramo še kraje za x_{0_1}, x_{0_2} in x_{0_3}:

x_{0_0}	x_{0_1}	x_{0_2}	x_{0_3}	x_{0_4}
0	1	1[1]		4
		2	1	
			2	
			3	

Znižamo omejitve, ki se tičejo skupine 0 ($q(3, 1) = 50 - 25 = 25, q(4, 2) = 50 - 25 = 25, q(5, 3) = 50 - 25 = 25$).

Nadaljujemo s skupino 1: zanjo imamo vektor x_1, $|x_1| = 4$, $x_{1_0} = 2$ in $x_{1_3} = 0$. Določiti moramo še kraje za x_{1_1} in x_{1_2}:

[1]Kraj je označen s sivo barvo, če ne ustreza.

x_{1_0}	x_{1_1}	x_{1_2}	x_{1_3}
2	1	1	0
		3	

Znižamo omejitve, ki se tičejo skupine 1 ($q(4,1) = 50-10 = 40, q(5,3) = 25-10 = 15$).

Nadaljujemo s skupino 2: zanjo imamo vektor x_2, $|x_2| = 4$, $x_{2_0} = 1$ in $x_{2_3} = 2$. Določiti moramo še kraje za x_{2_1} in x_{2_2}:

x_{2_0}	x_{2_1}	x_{2_2}	x_{2_3}
1	0	0	2
		3	

Znižamo omejitve, ki se tičejo skupine 2 ($q(9,0) = 50-11 = 39, q(10,3) = 50-11 = 39$).

Ker smo našli rešitev za vse skupine, izračunamo ceno. Dobimo vrednost: cena = 28660. To je trenutno najboljša rešitev.

Sedaj zvišamo omejitve, ki se tičejo skupine 2 ($q(9,0) = 39+11 = 50, q(10,3) = 39+11 = 50$), in iščemo naslednjo rešitev za skupino 2:

x_{2_0}	x_{2_1}	x_{2_2}	x_{2_3}
1	0	3	2
		$\hookleftarrow$[2]	
	3	0	
		3	
		$\hookleftarrow$	
	$\hookleftarrow$		

Ker ni nove rešitve za skupino 2, se vrnemo eno skupino nazaj in iščemo novo rešitev za skupino 1: zvišamo omejitve, ki se tičejo skupine 1, in iščemo naslednjo rešitev za skupino 1:

x_{1_0}	x_{1_1}	x_{1_2}	x_{1_3}
2	1	3	0
		$\hookleftarrow$	
	3	1	

Znižamo omejitve, ki se tičejo skupine 1.

Nadaljujemo s skupino 2:

x_{2_0}	x_{2_1}	x_{2_2}	x_{2_3}
1	0	0	2
		3	

Znižamo omejitve, ki se tičejo skupine 2.

Ker smo našli rešitev za vse skupine, izračunamo ceno. Dobimo vrednost: cena = 26010. To je trenutno najboljša rešitev.

[2] Puščica $\hookleftarrow$ pomeni, da smo prišli do zadnjega možnega kraja, in ker ne ustreza, gremo za eno komponento nazaj.

Sedaj zvišamo omejitve, ki se tičejo skupine 2, in iščemo naslednjo rešitev za skupino 2:

x_{2_0}	x_{2_1}	x_{2_2}	x_{2_3}
1	0	3	2
		$\hookleftarrow$	
	3	0	
		3	
		$\hookleftarrow$	
	$\hookleftarrow$		

Ker ni nove rešitve za skupino 2, se vrnemo eno skupino nazaj in iščemo novo rešitev za skupino 1: zvišamo omejitve, ki se tičejo skupine 1, in iščemo naslednjo rešitev za skupino 1:

x_{1_0}	x_{1_1}	x_{1_2}	x_{1_3}
2	3	1	0
		3	
		$\hookleftarrow$	
	$\hookleftarrow$		

Ker ni nove rešitve za skupino 1, se vrnemo eno skupino nazaj in iščemo novo rešitev za skupino 0: zvišamo omejitve, ki se tičejo skupine 0, in iščemo naslednjo rešitev za skupino 0:

x_{0_0}	x_{0_1}	x_{0_2}	x_{0_3}	x_{0_4}
0	1	2	3	4
			$\hookleftarrow$	
		3	1	
			2	
			3	
			$\hookleftarrow$	
		$\hookleftarrow$		
	2			
	3	1	1	
			2	
			3	
			$\hookleftarrow$	
		2	1	

Znižamo omejitve, ki se tičejo skupine 0.

Nadaljujemo s skupino 1:

x_{1_0}	x_{1_1}	x_{1_2}	x_{1_3}
2	1	1	0
		3	

Znižamo omejitve, ki se tičejo skupine 1.

Nadaljujemo s skupino 2:

x_{2_0}	x_{2_1}	x_{2_2}	x_{2_3}
1	0	0	2
		3	

Znižamo omejitve, ki se tičejo skupine 2.

Ker smo našli rešitev za vse skupine, izračunamo ceno. Dobimo vrednost: cena = 28660. To **ni** trenutno najboljša rešitev.

Sedaj zvišamo omejitve, ki se tičejo skupine 2, in iščemo naslednjo rešitev za skupino 2:

x_{2_0}	x_{2_1}	x_{2_2}	x_{2_3}
1	0	3	2
		$\hookleftarrow$	
	3	0	
		3	
		$\hookleftarrow$	
	$\hookleftarrow$		

Ker ni nove rešitve za skupino 2, se vrnemo eno skupino nazaj in iščemo novo rešitev za skupino 1: zvišamo omejitve, ki se tičejo skupine 1, in iščemo naslednjo rešitev za skupino 1:

x_{1_0}	x_{1_1}	x_{1_2}	x_{1_3}
2	1	3	0
		$\hookleftarrow$	
	3	1	

Znižamo omejitve, ki se tičejo skupine 1.

Nadaljujemo s skupino 2:

x_{2_0}	x_{2_1}	x_{2_2}	x_{2_3}
1	0	0	2
		3	

Znižamo omejitve, ki se tičejo skupine 2.

Ker smo našli rešitev za vse skupine, izračunamo ceno. Dobimo vrednost: cena = 23610. To je trenutno najboljša rešitev.

Sedaj zvišamo omejitve, ki se tičejo skupine 2, in iščemo naslednjo rešitev za skupino 2:

x_{2_0}	x_{2_1}	x_{2_2}	x_{2_3}
1	0	3	2
		$\hookleftarrow$	
	3	0	
		3	
		$\hookleftarrow$	
	$\hookleftarrow$		

Ker ni nove rešitve za skupino 2, se vrnemo eno skupino nazaj in iščemo novo rešitev

za skupino 1: zvišamo omejitve, ki se tičejo skupine 1, in iščemo naslednjo rešitev za skupino 1:

x_{1_0}	x_{1_1}	x_{1_2}	x_{1_3}
2	3	1	0
		3	
		↩	
	↩		

Ker ni nove rešitve za skupino 1, se vrnemo eno skupino nazaj in iščemo novo rešitev za skupino 0: zvišamo omejitve, ki se tičejo skupine 0, in iščemo naslednjo rešitev za skupino 0:

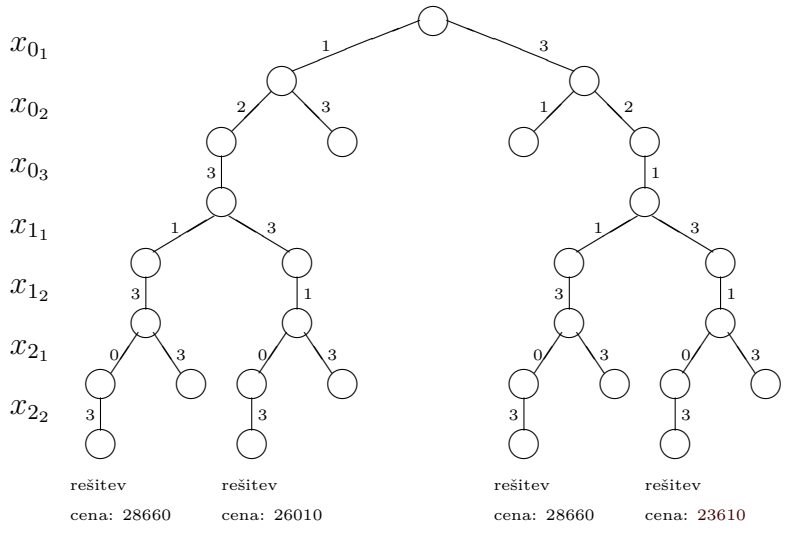
x_{0_0}	x_{0_1}	x_{0_2}	x_{0_3}	x_{0_4}
0	3	2	1	4
			2	
			3	
			↩	
		3		
		↩		
	↩			

Ker ni rešitve za skupino 0, končamo.

Najboljša rešitev je rešitev s ceno 23610. Ta rešitev je:

skupina i	x_{i_0}	x_{i_1}	x_{i_2}	x_{i_3}	x_{i_4}
0	0	3	2	1	4
1	2	3	1	0	–
2	1	0	3	2	–

Oglejmo si drevo stanj za ta primer:



4.3 Dinamično programiranje

Dinamično programiranje [2, 9, 10] je način reševanja problemov, ki sestojijo iz podproblemov, končna rešitev pa je sestavljena iz delnih rešitev. Ko na tekočem koraku ugotovimo, da delna rešitev neke rešitve ne vodi k cilju, to delno rešitev zavržemo. Iz splošne narave problema izpeljemo pravila, s katerimi krčimo število potencialnih rešitev. Dinamično programiranje temelji na pravilu optimalnosti, ki od problema, ki ga rešujemo, zahteva, da mora vsako podzaporedje zaporedja odločitev, ki generira optimalno rešitev, biti tudi optimalno.

Kako dela algoritem, ki uporablja dinamično programiranje:

Pri sestavljanju algoritma si pomagamo z dinamičnim algoritmom za reševanje problema trgovskega potnika (glej [2]). Pri našem algoritmu dela dinamično povrsti po dnevih. Zakaj po dnevih in ne povrsti po skupinah, tako kot pri sestopanju? Če bi se namreč lotili povrsti po skupinah, bi se lahko za neko skupino kot najboljše zaporedje krajev pokazalo takrat, ko bi delali za to skupino, določeno zaporedje, a ko bi iskali zaporedje za neko drugo skupino, se izkaže, da bi bilo za prejšnjo boljše drugo zaporedje, npr. zaradi prekrivanja krajev.

Najprej preverimo, ali lahko dodamo vse začetne in končne kraje, in če jih lahko, potem tam, kjer skupine imajo v katerem od teh krajev ogled, od $q(d_{iA}, A), A \in \{Z_i, K_i\}$, odštejemo q_i. Če začetnih in končnih krajev ne moremo dodati, končamo (ni rešitve).

Nato za vsako skupino pogledamo, kdaj ima prvi premik. Iz tega dobimo dan d, ko se zgodi "v splošnem" prvi premik.

Kako deluje po dnevih:

Najprej posebej pogledamo za dan d, katere vse skupine imajo premik. Za vsako od teh skupin pogledamo, v katere vse notranje kraje gre, in naredimo vse možne kombinacije tako, da za vsako skupino izberemo en kraj.

Nato za vsako od teh kombinacij izračunamo ceno (cena leta iz začetnega kraja v ta kraj, pomnožena s številom potnikov in pa stroški za vsak kraj, ki se ga obišče in še ni nobena skupina od prej prišla tja). Če kakega leta ni ali ne more priti toliko potnikov v kraj (ni dovolj kapacitet), zbrišemo to kombinacijo.

Za vsako od teh kombinacij izračunamo za vsako od teh skupin, kdaj ima naslednji premik.

Od vseh kombinacij in vseh skupin, ki še niso imele premika, izračunamo, kdaj je prvi naslednji premik, ta dan poimenujemo dan d.

Vsaka od teh kombinacij je ena podrešitev.

Nato delamo, dokler ne pridemo do zadnjega dneva:

Od skupin, ki še niso imele premika, poiščemo tiste, ki se premaknejo na dan d. Za vsako od teh skupin pogledamo, v katere vse notranje kraje gre in naredimo vse možne kombinacije tako, da za vsako skupino izberemo en kraj.

Če je vsaj ena taka skupina, potem:

Iz vseh "še neuporabljenih" podrešitev naredimo vse možne različne kombinacije, tako da dodamo k tistim skupinam v podrešitvi, ki se premaknejo ta dan, še en kraj in te kombinacije skombiniramo z "novimi" skupinami. Te podrešitve postanejo "uporabljene".

Tako dobimo vse možne kombinacije premikov za ta dan.

Nato za vsako od teh kombinacij izračunamo ceno, to je minimum vsote, sestavljene iz:

- cene ene od ustreznih podrešitev
- za vsako skupino te podrešitve, ki se premika, cene leta iz prejšnjega kraja v ta kraj, pomnožene s številom potnikov
- za nove skupine cene leta iz začetnega kraja v ta kraj, pomnožene s številom potnikov
- stroškov za vsak kraj, ki se ga obišče in ni še nobene skupine od prej tam

Če za kombinacijo ni dopustne rešitve, zbrišemo to kombinacijo.

Za vsako od teh kombinacij izračunamo za vsako od teh skupin, kdaj ima naslednji premik.

Vsaka kombinacija, ki ima rešitev, je nova podrešitev.

Za vsako od teh kombinacij izračunamo za vsako od teh skupin, kdaj ima naslednji premik.

Če ni nobene take skupine, potem:

Od vseh "še neuporabljenih" podrešitev, kjer ima vsaj ena skupina ta dan premik, naredimo vse možne različne kombinacije, tako da dodamo k tistim skupinam v določeni podrešitvi, ki se premaknejo ta dan, še en kraj. Te podrešitve postanejo "uporabljene".

Tako dobimo vse možne kombinacije premikov za ta dan.

Nato za vsako od teh kombinacij izračunamo ceno, to je minimum vsote, sestavljene iz:

- cene ene od ustreznih podrešitev
- za vsako skupino te podrešitve, ki se premika, cene leta iz prejšnjega kraja v ta kraj, pomnožene s številom potnikov

- stroškov za vsak kraj, ki se ga obišče in ni še nobene skupine od prej tam

Če za kombinacijo ni dopustne rešitve, zbrišemo to kombinacijo.

Za vsako od teh kombinacij izračunamo za vsako od teh skupin, kdaj ima naslednji premik.

Vsaka kombinacija, ki ima rešitev, je nova podrešitev.

Za vsako od teh kombinacij izračunamo za vsako od teh skupin, kdaj ima naslednji premik.

Od vseh kombinacij in vseh skupin, ki še niso imele premika, izračunamo, kdaj je prvi naslednji premik, sedaj ta dan poimenujemo dan d.

Za zadnji dan dobimo le eno podrešitev, to je končna rešitev.

Ker je pri dinamičnem programiranju končna (optimalna) rešitev sestavljena iz delnih rešitev, potrebujemo razred, v katerem imamo shranjene vse podatke o določeni delni rešitvi. Ta razred poimenujemo C. Podatki v tem razredu so:

- dan d, torej dan, za katerega iščemo optimalno delno rešitev,
- skupine, ki se na ta dan premaknejo v drug kraj,
- katere vse kraje je vsaka skupina, ki je že na potovanju, že obiskala, vključno z današnjim,
- kdaj skupina zapusti kraj, v katerega ta dan pride,
- vsi stroški, ki so nastali do tega dne (vključno s tem dnem) - cene letalskih kart in stroški ogledov v krajih,
- ko za ta C najdemo najboljšo rešitev, moramo shraniti, iz katere delne rešitve (torej nekega prejšnjega C-ja) smo dobili to rešitev,
- katere vse prejšnje delne rešitve so ustrezne za to delno rešitev, da ko računamo minimum od C, vemo, katere vse podrešitve gledati.

Pa si poglejmo na enakem primeru, kot smo ga naredili pri sestopanju, kako dela algoritem, ki uporablja dinamično programiranje.

4.3.1 Primer

Najprej preverimo, ali lahko dodamo začetne in končne kraje. Ker jih lahko dodamo, spremenimo $q(d, A)$ kot pri sestopanju in izračunamo še ceno za bivanje v teh krajih. Ta cena je enaka $4 \cdot 2400 = 9600$.

Nato za vsako skupino izračunamo, kateri dan se zgodi prvi premik:

skupina i	dan, ko se zgodi prvi premik
0	3
1	4
2	9

Izračunamo prvi dan, ko se zgodi premik: $\min\{3, 4, 9\} = 3$. Pogledamo, katere skupine imajo prvi premik ta dan, to je skupina 0. Za ta dan izračunamo posebej:

dan 3 : $\boxed{1}$ $C(\{\{0,1\}\},\{0\}) = c_{01} \cdot 25 + 2400 = 2900$ naslednji dnevi : $\{4\}$

$C(\{\{0,2\}\},\{0\}) = c_{02} \cdot 25 + \infty = \infty$ zbrišemo

$\boxed{2}$ $C(\{\{0,3\}\},\{0\}) = c_{03} \cdot 25 + 2400 = 3400$ naslednji dnevi : $\{4\}$

Izračunamo konec: $\max\{\text{prihod v zadnji kraj za vsako skupino}\} = 14$

Dokler dan ≤ 14 delamo na enak način:

Izračunamo naslednji dan, ko se zgodi premik: $\min\{4, 9, 4, 4\} = 4$. Pogledamo, katere skupine imajo prvi premik ta dan, to je skupina 1.

dan 4 : $\boxed{3}$ $C(\{\{0,1,2\},\{2,1\}\},\{0,1\}) =$
$\min(\boxed{1} + c_{12} \cdot 25 + c_{21} \cdot 10 + 2400 + 2400) = 8750$
naslednji dnevi : $\{5, 5\}$

$\boxed{4}$ $C(\{\{0,1,2\},\{2,3\}\},\{0,1\}) =$
$\min(\boxed{1} + c_{12} \cdot 25 + c_{23} \cdot 10 + 2400 + 0) = 6300$
naslednji dnevi : $\{5, 6\}$

$\boxed{5}$ $C(\{\{0,1,3\},\{2,1\}\},\{0,1\}) =$
$\min(\boxed{1} + c_{13} \cdot 25 + c_{21} \cdot 10 + 0 + 2400) = 5975$
naslednji dnevi : $\{5, 5\}$

$C(\{\{0,1,3\},\{2,3\}\},\{0,1\}) =$
$\min(\boxed{1} + c_{13} \cdot 25 + c_{23} \cdot 10 + \infty) = \infty$
zbrišemo

$\boxed{6}$ $C(\{\{0,3,1\},\{2,1\}\},\{0,1\}) =$
$\min(\boxed{2} + c_{31} \cdot 25 + c_{21} \cdot 10 + 2400) = 6475$
naslednji dnevi : $\{5, 5\}$

$\boxed{7}$ $C(\{\{0,3,1\},\{2,3\}\},\{0,1\}) =$
$\min(\boxed{2} + c_{31} \cdot 25 + c_{23} \cdot 10 + 2400 + 0) = 6425$
naslednji dnevi : $\{5,6\}$

$\boxed{8}$ $C(\{\{0,3,2\},\{2,1\}\},\{0,1\}) =$
$\min(\boxed{2} + c_{32} \cdot 25 + c_{21} \cdot 10 + 2400 + 2400) = 9125$
naslednji dnevi : $\{5,5\}$

$\boxed{9}$ $C(\{\{0,3,2\},\{2,3\}\},\{0,1\}) =$
$\min(\boxed{2} + c_{32} \cdot 25 + c_{23} \cdot 10 + 2400 + 0) = 6675$
naslednji dnevi : $\{5,6\}$

Izračunamo naslednji dan, ko se zgodi premik:

$$\min\{9,5,5,5,6,5,5,5,5,5,6,5,5,5,6\} = 5.$$

Pogledamo, katere skupine imajo prvi premik ta dan, ni nobene take skupine.

dan 5 : $\boxed{10}$ $C(\{\{0,1,2,3\},\{2,1,3\}\},\{0,1\}) =$
$\min(\boxed{3} + c_{23} \cdot 25 + c_{13} \cdot 10 + 2400) = 11925$
naslednji dnevi : $\{6,7\}$

$\boxed{11}$ $C(\{\{0,1,2,3\},\{2,3\}\},\{0\}) =$
$\min(\boxed{4} + c_{23} \cdot 25 + 2400) = 9325$
naslednji dnevi : $\{6,6\}$

$\boxed{12}$ $C(\{\{0,1,3,2\},\{2,1,3\}\},\{0,1\}) =$
$\min(\boxed{5} + c_{32} \cdot 25 + c_{13} \cdot 10 + 2400 + 2400,$
$\boxed{6} + c_{12} \cdot 25 + c_{13} \cdot 10 + 2400 + 2400) =$
$\min(11550, 12175) = 11550$
naslednji dnevi : $\{6,7\}$

$\boxed{13}$ $C(\{\{0,1,3,2\},\{2,3\}\},\{0\}) =$
$\min(\boxed{7} + c_{12} \cdot 25 + 2400) = 9575$
naslednji dnevi : $\{6,6\}$

$\boxed{14}$ $C(\{\{0,3,2,1\},\{2,1,3\}\},\{0,1\}) =$
$\min(\boxed{8} + c_{21} \cdot 25 + c_{13} \cdot 10 + 0 + 2400) = 12425$
naslednji dnevi : $\{6,7\}$

$\boxed{15}$ $C(\{\{0,3,2,1\},\{2,3\}\},\{0\}) =$
$\min(\boxed{9} + c_{21} \cdot 25 + 0) = 7425$
naslednji dnevi : $\{6,6\}$

Izračunamo naslednji dan, ko se zgodi premik: $\min\{9,6,7,6,6,6,7,6,6,6,7,6,6\} = 6$. Pogledamo, katere skupine imajo prvi premik ta dan, ni nobene take skupine.

dan 6 : $\boxed{16}$ $C(\{\{0,1,2,3,4\},\{2,1,3\}\},\{0\}) =$
$\min(\boxed{10} + c_{34} \cdot 25,$
$\boxed{12} + c_{24} \cdot 25,$
$\boxed{14} + c_{14} \cdot 25) =$
$\min(12925, \infty, 12925) = 12925$
naslednji dnevi : $\{\infty, 7\}$

$\boxed{17}$ $C(\{\{0,1,2,3,4\},\{2,3,1\}\},\{0,1\}) =$
$\min(\boxed{11} + c_{34} \cdot 25 + c_{31} \cdot 10 + 0,$
$\boxed{13} + c_{24} \cdot 25 + c_{31} \cdot 10 + 0,$
$\boxed{15} + c_{14} \cdot 25 + c_{31} \cdot 10 + 0) =$
$\min(10475, \infty, 8075) = 8075$
naslednji dnevi : $\{\infty, 7\}$

Izračunamo naslednji dan, ko se zgodi premik: $\min\{9, \infty, 7, \infty, 7\} = 7$. Pogledamo, katere skupine imajo prvi premik ta dan, ni nobene take skupine.

dan 7 : $\boxed{18}$ $C(\{\{0,1,2,3,4\},\{2,1,3,0\}\},\{1\}) =$
$\min(\boxed{16} + c_{30} \cdot 10,$
$\boxed{17} + c_{10} \cdot 10) =$
$\min(13325, 8275) = 8275$
naslednji dnevi : $\{\infty, \infty\}$

Izračunamo naslednji dan, ko se zgodi premik: $\min\{9, \infty, \infty, \} = 9$. Pogledamo, katere skupine imajo prvi premik ta dan, to je skupina 2.

dan 9 : $\boxed{19}$ $C(\{\{0,1,2,3,4\},\{2,1,3,0\},\{1,0\}\},\{2\}) =$
$\min(\boxed{18} + c_{10} \cdot 11 + 2400) = 10895$
naslednji dnevi : $\{\infty, \infty, 10\}$

$\boxed{20}$ $C(\{\{0,1,2,3,4\},\{2,1,3,0\},\{1,3\}\},\{2\}) =$
$\min(\boxed{18} + c_{13} \cdot 11 + 2400) = 10840$
naslednji dnevi : $\{\infty, \infty, 13\}$

Izračunamo naslednji dan, ko se zgodi premik: $\min\{\infty, \infty, 10, \infty, \infty, 13\} = 10$. Pogledamo, katere skupine imajo prvi premik ta dan, ni nobene take skupine.

dan 10 : $\boxed{21}$ $C(\{\{0,1,2,3,4\},\{2,1,3,0\},\{1,0,3\}\},\{2\}) =$
$\min(\boxed{19} + c_{03} \cdot 11 + 2400) = 13735$
naslednji dnevi : $\{\infty, \infty, 14\}$

Izračunamo naslednji dan, ko se zgodi premik: $\min\{\infty, \infty, 13, \infty, \infty, 14\} = 13$. Pogledamo, katere skupine imajo prvi premik ta dan, ni nobene take skupine.

dan 13 : $\boxed{22}$ $C(\{\{0,1,2,3,4\},\{2,1,3,0\},\{1,3,0\}\},\{2\}) =$
$\min(\boxed{20} + c_{30} \cdot 11 + 2400) = 13680$
naslednji dnevi : $\{\infty, \infty, 14\}$

Izračunamo naslednji dan, ko se zgodi premik: $\min\{\infty, \infty, 14, \infty, \infty, 14\} = 14$. Pogledamo, katere skupine imajo prvi premik ta dan, ni nobene take skupine.

dan 14 : $\boxed{23}$ $C(\{\{0,1,2,3,4\},\{2,1,3,0\},\{1,0,3,2\}\},\{2\}) =$
$\min(\boxed{21} + c_{32} \cdot 11,$
$\boxed{22} + c_{02} \cdot 11) =$
$\min(14010, \infty) = 14010$
naslednji dnevi : $\{\infty, \infty, \infty\}$

Končamo.

Končna vsota je enaka $14010 + 9600 = 23610$.

Pogledamo nazaj, katere smo izbrali, da dobimo končno rešitev:

katera\skupina	0	1	2
$\boxed{23}$			2
$\boxed{21}$			3
$\boxed{19}$			0
$\boxed{18}$		0	
$\boxed{17}$	4	1	
$\boxed{15}$	1	3	
$\boxed{9}$	2		
$\boxed{2}$	3		

Iz tega dobimo zaporedje potovanja (rešitev beremo od spodaj navzgor in na začetek dodamo še začetni kraj):

skupina 0	0	3	2	1	4
skupina 1	2	3	1	0	
skupina 2	1	0	3	2	

Oziroma, če si ogledamo rešitev po dnevih (kraj je napisan s pomanjšano pisavo, če skupina biva ta dan v njem, a v njem nima ogleda, in z rdečo pisavo, če je začetni ali končni kraj, skupina pa v njem nima ogleda):

dan	2	3	4	5	6	7	8	9	10	11	12	13	14
skupina 0	0	3	2	1	4								
skupina 1		2	3	3	1	0							
skupina 2					1	1	1	0	3	3	3	3	2

4.4 Približevanje k optimalni rešitvi z 2-zamenami

2-zamena je metoda lokalne optimizacije [11]. V našem primeru jo definiramo tako: podano imamo neko dopustno rešitev problema, za vsaj eno skupino izberemo 2 notranja kraja in ju zamenjamo (s tem se lahko spremenijo tudi dnevi prihodov za kraje, ki so med tema dvema krajema). Nato preverimo, ali je taka rešitev dopustna, in če je, izračunamo novo končno ceno. Če je prvotna cena vsaj tako dobra kot cene, ki jih dobimo z 2-zamenami vseh možnih zamenjav, smo našli lokalni minimum glede na z 2-zamenami določeno okolico rešitve.

Kako dela algoritem, ki uporablja 2-zamene:

Najprej s sestopanjem poiščemo prvo dopustno rešitev in izračunamo njeno ceno. Pregledamo vse možne kombinacije dveh notranjih krajev za vsako skupino in naredimo vse možne kombinacije iz teh kombinacij, tako da iz vsake skupine vzamemo eno kombinacijo.

Potem delamo, dokler ne najdemo lokalno najboljše rešitve:

Imamo podane neke $q(d, A)$

Za vsako od teh kombinacij:

Za vsako skupino poiščemo, na katerih indeksih sta ta dva kraja iz kombinacije. Označimo indeksa z i in j.

Odštejemo cene letalskih kart, pomnožene s številom potnikov skupine, med kraji na indeksih $(i-1, i), (i, i+1), (j-1, j)$ ter $(j, j+1)$, če kraja nista sosednja, oz. na indeksih $(i-1, i), (i, j)$ ter $(j, j+1)$, če kraja sta sosednja.

Za vsako skupino zvišamo omejitve v vseh krajih, ki so trenutno na indeksih med i in j, in če se kje zgodi, da pri zviševanju pridemo do tega, da v kraju ni nobenega potnika, torej da $q(d, A) = p(A)$, odštejemo za dan d, kraj A, konstanto.

Nato za vsako skupino zamenjamo kraja na indeksih i in j, za kraje med njima in za kraj na indeksu j izračunamo dneve prihodov vanje.

Za vsako skupino prištejemo cene letalskih kart, pomnožene s številom potnikov, med kraji na indeksih $(i-1, i), (i, i+1), (j-1, j)$ ter $(j, j+1)$, če kraja nista sosednja, oz. na indeksih $(i-1, i), (i, j)$ ter $(j, j+1)$, če kraja sta sosednja.

Za vsako skupino znižamo omejitve v vseh krajih, ki so trenutno na indeksih med i in j, in če se kje zgodi, da preden znižamo, je $q(d, A) = p(A)$, prištejemo za dan d, kraj A, konstanto.

Če med kakšnima krajema na indeksih $(i-1, i), (i, i+1), (j-1, j)$ in $(j, j+1)$ ni leta, potem končamo za to kombinacijo (zanjo ni rešitve).

Če pri zniževanju omejitev pridemo kje do $q(d, A) < 0$, potem končamo za to kombinacijo (zanjo ni rešitve).

Če je ta cena cenejša od trenutno najcenejše, nastavi najcenejšo ceno na to ceno in trenutno najboljšo rešitev na to rešitev ter še shrani $q(d, A)$ za to rešitev.

Tako dobimo najboljšo rešitev kombinacij, $q(d, A)$ za to rešitev in ceno za to rešitev.

Če je ta rešitev lokalno najboljša, jo shrani kot trenutno najboljšo rešitev in ponovi spet za vse kombinacije ...

Sicer končaj. Lokalno najboljša rešitev je trenutno najboljša rešitev.

4.5 Rezultati in komentarji

Meritve so bile izvedene na računalniku, ki ima vgrajen procesor Intel Pentium s frekvenco 2,80 GHz in ima 2 GB pomnilnika. Vhodni podatki so bili izbrani naključno. Rezultate bomo predstavili v treh tabelah. V prvi bo čas, ki ga je za rešitev potreboval algoritem s pomočjo sestopanja, v drugi bo čas, ki ga je za rešitev potreboval algoritem, ki uporablja dinamično programiranje, v tretji pa bo rezultat, s kolikšno natančnostjo se 2-zamene približajo najcenejši rešitvi. V prvih dveh tabelah je zelene barve hitrejša rešitev. Po vrsticah imamo število skupin, po stolpcih pa število notranjih krajev.

Sestopanje:

	2	3	4	5
1	< 0.1 s	< 0.1 s	< 0.1 s	< 0.1 s
2	< 0.1 s	< 0.1 s	< 0.1 s	0.3 s
3	< 0.1 s	< 0.1 s	0.1 s	1 min 7.2 s
4	< 0.1 s	< 0.1 s	7.3 s	9 min 32.1 s
5	< 0.1 s	0.3 s	6 min 28.1 s	
6	< 0.1 s	1.2 s		
7	< 0.1 s	0.8 s		
8	< 0.1 s	45.6 s		
9	< 0.1 s	2 min 4.8 s		
10	< 0.1 s	1 h 4 min 29.5 s		
12	0.1 s	11 h 48 min 13.9 s		
14	0.1 s			
16	1 min 14.3 s			
18	52.2 s			

	6	7	8	9
1	< 0.1 s	0.1 s	1.6 s	29.0 s
2	7.9 s	27 min 42.8 s	49 h 44 min 40.5 s	> 90 h

Dinamično:

	2	3	4	5
1	< 0.1 s	0.1 s	< 0.1 s	< 0.1 s
2	< 0.1 s	< 0.1 s	< 0.1 s	0.3 s
3	< 0.1 s	0.1 s	0.3 s	1 min 17.5 s
4	0.1 s	0.3 s	2.7 s	38 min 22.8 s
5	0.2 s	0.2 s	17 min 56.7 s	
6	< 0.1 s	27.5 s		
7	< 0.1 s	30.6 s		
8	< 0.1 s	5 min 58.9 s		
9	0.3 s	25.2 s		
10	0.2 s	22 min 57.3 s		
12	0.1 s	55 h 27 min 16.7 s		
14	0.1 s			
16	0.1 s			
18	0.2 s			

	6	7	8	9
1	0.1 s	0.1 s	< 0.1 s	0.3 s
2	2.1 s	1 min 32.0 s	30 min 5.6 s	6 h 11 min 32.5 s

2-zamene:

	2	3	4	5	6	7	8	9
1	1,00	1,00	1,05	1,00	1,57	1,44	2,30	2,57
2	1,00	1,11	1,25	1,00	1,30	1,25	1,11	1,30
3	1,00	1,20	1,00	1,07				
4	1,00	1,07	1,00	1,50				
5	1,00	1,00	1,01					
6	1,00	1,02						
7	1,00	1,19						
8	1,00	1,00						
9	1,00							
10	1,00							
12	1,00							
14	1,00							

Vidimo torej, da se v povprečju pri malo krajih za boljšega izkaže algoritem s sestopanjem, če pa je notranjih krajev veliko, je boljši algoritem, ki uporablja dinamično programiranje. Seveda so časi odvisni od števila dopustnih rešitev in pri dinamičnem še od časovnih intervalov potovanja.

Komentarji:

Pri rezultatih 2-zamen so rezultati odvisni od tega, kako so povrsti podani notranji kraji pri vhodnih podatkih. Namreč, če enake vhodne notranje kraje podamo v drugačnem zaporedju, dobimo drugačno začetno rešitev s sestopanjem in začnemo iskati rešitev z 2-zameno z drugačno "prvo" rešitvijo. Zato lahko dobimo tudi končni rezultat drugačen.

Algoritme bi lahko izboljšali z raznimi spremembami, nekatere od njih so:

- Algoritme lahko dopolnimo še s tem, da če kakšna skupina obišče le en notranji kraj, obravnavamo to podobno kot za začetni in končni kraj v splošnem. Celo potovanje te skupine je namreč enolično določeno, zato pri algoritmih vse podatke za to skupino dokončno nastavimo na začetku (tako kot dodamo za ostale skupine začetni in končni kraj, za to skupino dodamo tudi notranjega). Potem v nadaljevanju ne preverjamo ničesar za to skupino.

- Če bi se potovanje vseh skupin, za katere iščemo razporeditev, lahko razdelilo na disjunktne časovne intervale, bi lahko iskali rešitev za vsak ta interval posebej, saj so neodvisni med sabo, s tem pa bi zelo skrajšali časovno zahtevnost.

- Pri sestopanju sedaj izračunamo ceno šele, ko najdemo dopustno rešitev. Namesto tega bi lahko ceno sproti računali, in če bi cena pri neki komponenti presegla trenutno najboljšo rešitev, bi šli za to komponento iskat naslednjo rešitev.

- Lahko bi uporabili kakšno drugo lokalno optimizacijo ali hevristiko za približevanje k optimalni rešitvi.

- Pri 2-zamenah sedaj na začetku shranimo vse možne kombinacije krajev, ki jih lahko zamenjamo in potem za vsako to kombinacijo pogledamo, kakšna pride cena, če na njej naredimo 2-zameno. Če bi se pojavil problem s prostorom zaradi preveč kombinacij, bi lahko kombinacije izbirali sproti.

Problem bi lahko dopolnili tako, da bi bile cene za vsak kraj za vsak dan drugačne, prav tako bi lahko bile cene letov vsak dan drugačne. Če bi problem dopolnili tako, bi v programu namesto konstante za ceno ogleda imeli tabelo cen ogledov glede na kraj in dan, prav tako bi namesto ene tabele cen letov imeli več tabel cen letov (za vsak dan eno) in bi bilo potrebno v algoritmih pri računanju cen dostopati do pravilnih podatkov.

Pri celoštevilskem linearnem programu bi se verjetno dalo še kakšne spremenljivke takoj določiti (jih ne računati z enačbami), ali pa za dneve, ko skupina ne potuje, in za kraje, v katere skupina ne gre, sploh ne bi imeli spremenljivk. Potem bi bilo treba spremeniti enačbo za računanje vektorja o in za računanje kriterijske funkcije.

Literatura

[1] Hilyard, J. in Teilhet, S. *C# Cookbook*. 2. izdaja. Sebastopol: O'Reilly, 2006.

[2] Kozak, J. *Podatkovne strukture in algoritmi*. 2. natis. Ljubljana: Društvo matematikov, fizikov in astronomov Slovenije, 1997.

Spletni viri:

[3] *NP-complete* (online).
http://en.wikipedia.org/wiki/NP-complete (dostop februar 2010).

[4] *Travelling salesman problem* (online).
http://en.wikipedia.org/wiki/Travelling_salesman_problem (dostop februar 2010).

[5] *Partition problem* (online).
http://en.wikipedia.org/wiki/Partition_problem (dostop februar 2010).

[6] *Linear programming* (online).
http://en.wikipedia.org/wiki/Linear_programming (dostop februar 2010).

[7] *Sestopanje* (online).
http://wiki.fmf.uni-lj.si/wiki/Sestopanje (dostop februar 2010).

[8] *Sestopanje in iskanje* (online).
http://rc.fmf.uni-lj.si/matija/predavanja/2003-2004/predavanja2003-4/P25/Sestopanje.ppt (dostop februar 2010).

[9] *Dynamic programming* (online).
http://en.wikipedia.org/wiki/Dynamic_programming (dostop februar 2010).

[10] *Dinamično programiranje* (online).
http://sl.wikipedia.org/wiki/Dinamično_programiranje (dostop februar 2010).

[11] *Local search (optimization)* (online).
http://en.wikipedia.org/wiki/Local_search_(optimization) (dostop februar 2010).