Univerza *v Ljubljani*
Fakulteta za *matematiko in fiziko*

Razporejanje prenosa podatkov po omrežju

Modelska analiza

avtor
Matic Debeljak

profesor
prof. dr. Simon Širca
asistent
doc. dr. Miha Mihovilovič

Ljubljana, 2020

1 Naloga

Za model omrežja vzemi $N \times N$ kvadratno mrežo, vsak rob pa ima naključno maksimalno hitrost povezave med 0 in 1. Vozlišča na zgornjem robu so internetni odjemalci, spodnji rob pa strežniki. V notranjih vozliščih velja 1. Kirchhoffov zakon. S pomočjo linearnega programiranja določi, kolikšne hitrosti prenosa imajo strežniki in odjemalci, ko je skupna hitrost prenosa največja. Ker gre za naključna omrežja, si oglej tudi statistično porazdelitev zanimivih količin.

2 Teorija

Linearno programiranje (LP), imenovano tudi linearna optimizacija, je metoda za dosego najboljšega rezultata (npr. največjega dobička ali najnižjih stroškov) v matematičnem modelu, katerega zahteve so predstavljene z linearnimi povezavami. Bolj formalno, linearno programiranje je tehnika za optimizacijo linearnega cenilne funkcije ob upoštevanju linearnih omejitev enakosti in neenakosti.

Vsaka neenakost določa polprostor, skupaj te tvorijo konveksni polieder. Cilj linearnega programiranja je maksimirati ali minimizirati linearno funkcijo, ki je določena na tem poliedru. Algoritem linearnega programiranja išče točko v poliederu, kjer ima ta funkcija največjo (ali najmanjšo) vrednost, če taka točka obstaja. To zapišemo kot:

$$\max_{\mathbf{x}} \mathbf{c}^T \mathbf{x}$$

pri čemer veljajo linearni pogoji:

$$\mathbf{A}\mathbf{x} \leq \mathbf{b}$$

in ne-negativnost spremenljivk:

$$\mathbf{x} \geq \mathbf{0}$$

kjer je $\mathbf{x}$ vektor neznank, $\mathbf{c}$ vektor koeficientov ciljne funkcije, $\mathbf{A}$ matrika koeficientov omejitev in $\mathbf{b}$ vektor omejitev.

Kot enostaven primer linearnega programiranja si lahko pogledamo primer podjetja, ki proizvaja dva izdelkam, kjer z $x_{1,2}$ označimo dnevno količino prodanih izdelkov. Seveda želimo maksimizirati profit.

$$z = 20x_1 + 12x_2$$

Izdelk 1 ima vrednost 20€, izdelek 2 pa 12€. Delavci lahko proizvedejo makismalno 50 kosov na dan:

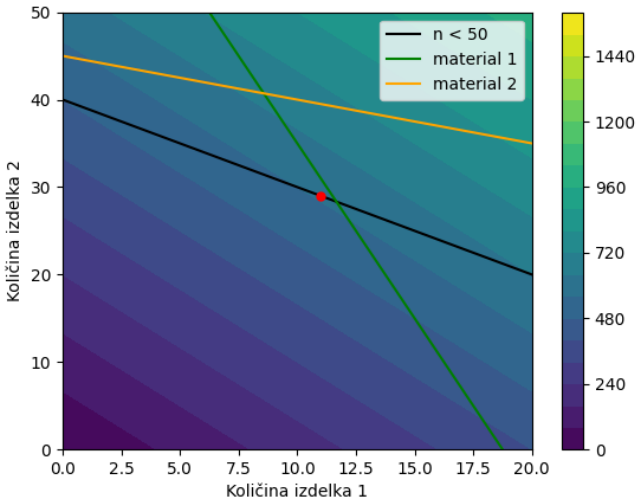
$$x1 + x2 < 40$$

prav tako imamo omejitve glede materiala, na dan lahko porabimo 150kg materiala A in 90kg materiala B. Za izdelek 1 potrebujemo 8kg materiala A in 1kg materiala B, za izdelek 2 pa 2kg materiala A in B.

$$8x_1 + 2x_2 < 150$$

$$x_1 + 2x_2 < 90$$

Opisani problem je dvo dimenizonalen (optimizirati moramo samo dve spremnljivki; izdelek 1 in izdelek 2), zato ga lahko vizuliziramo (slika 1).

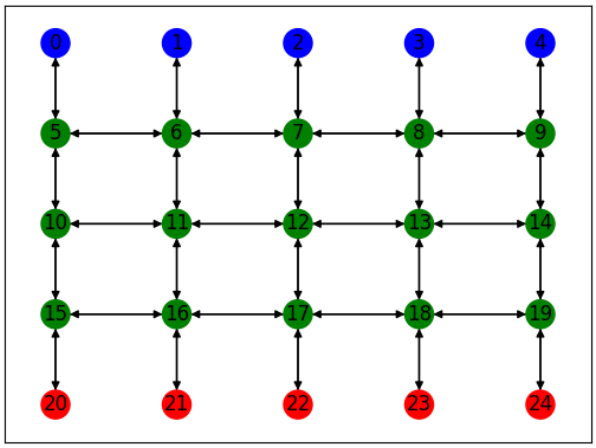


Slika 1: Rešitev enostavnega linearnega problema.

Na sliki 1, vidimo, da je optimalna rešitev probelma 11 izdelkov A in 29 izdelkov B, maksimalni profit pa 568 €. Enako rešitev dobimo tudi v pythonu z uporabo knjižjice pulp, ki za nas reši probleme linearnega programiranja. Lahko si predstavljamo, da ko dodamo več spremljivk problemov linearnega programiranja ne bomo morali več reševati "na roko"ali z vizualizacijo, zato nam bo še kako prav prišla uporaba računalnika.

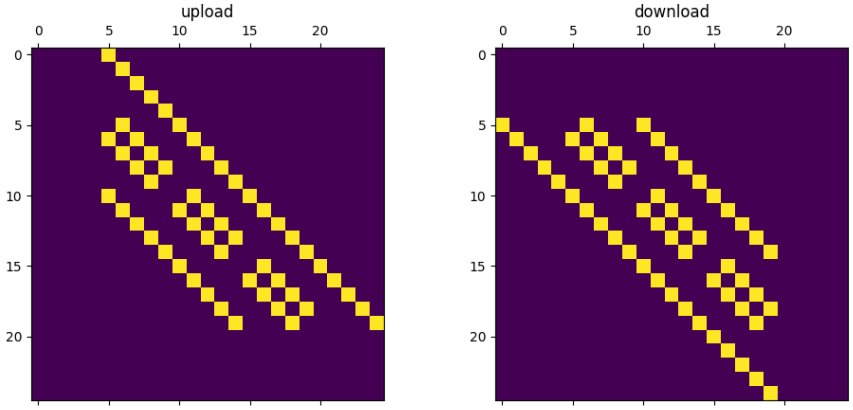
3 Internetno omrežje

Internetno omrežje sem modeliral z $N \times N$ kvadratno mrežo, v kateri so povezani samo najbližje sosedje. Zgornja vozlišča predstavljajo odjemalce, spodnja vozlišča pa strežniki (slika 2). Povezani so vsi najbližji sosedi razen odjemalci med sabo in strežniki med sabo.



Slika 2: Primer 5×5 omrežja, z modro barvo so označeni odjemalci, z rdečo barvo pa strežniki, ostala vozlišča (za katera velja 1. Kirchhoffov zakon) so obarvana zeleno. Dovoljene povezave so označene s črnimi puščicami.

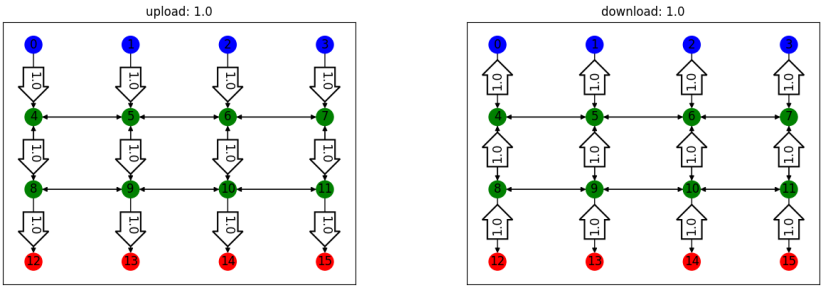
Internetno hitrost sem podobno kot v resničnem življenu razdelil na "upload" (smer od uporabnika do strežnika) in "download" (smer od strežnika do uporabnika). Na sliki 3 lahko vidimo, kako se razlikujejo dovoljene povezave za ta primera. Za primer upload vidimo, da uporabniki ne smejo sprejemati podatkov, strežniki pa jih ne smejo pošiljati. Pri downloadu je ravno obranto, torej strežniki lahko podatke le pošiljajo, uporabniki pa jih lahko le prejemjo. Rumen barva na mestu i, j torej pomeni, da lahko podatki tečejo iz vozlišča i v vozlišče j ne pa nujno obratno. kot primer lahko vzamemo matriko X za upload, $X_{0,5}$ je obarvan rumeno, kar pomeni da podatki lahko tečejo od uporabnika na vozlišče, medtem ko je $X_{5,0}$ vijolčne barve, torej ta povezava ni omogočena (uporabnik ne mora sprejmati podatkov iz ogliščа med uploadom).



Slika 3: Matrika dovoljeni povezav za ”upload” in ”download”. Z rumeno barvo so označene dovoljene povezave.

3.1 Popolno omrežje

Poglejmo si primer ”popolnega omrežja”, za katerega velja da so aktivirane vse povezave in vse omogočajo makimalno hitrost povezave (1). Ta problem je tudi precej enostaven za analizo in ga lahko rešimo sami, brez uporabe računalnika. Precej očitno je, da bojo podatki potovali direktno od uporabnikov do strežnikov in nazaj. Ter bo tako hitrost ”uploada” in ”downloada” za vsakega uporabnika in strežnik enaka 1. Poglejmo ali se s tem strinja tudi program. Kot cilnjo fukncijo sem maksimiral vsoto vseh hitrosti na strežnikih in uporabnikih. Seveda sem jo normiral tako, da je v popolnem omrežju enaka 1.

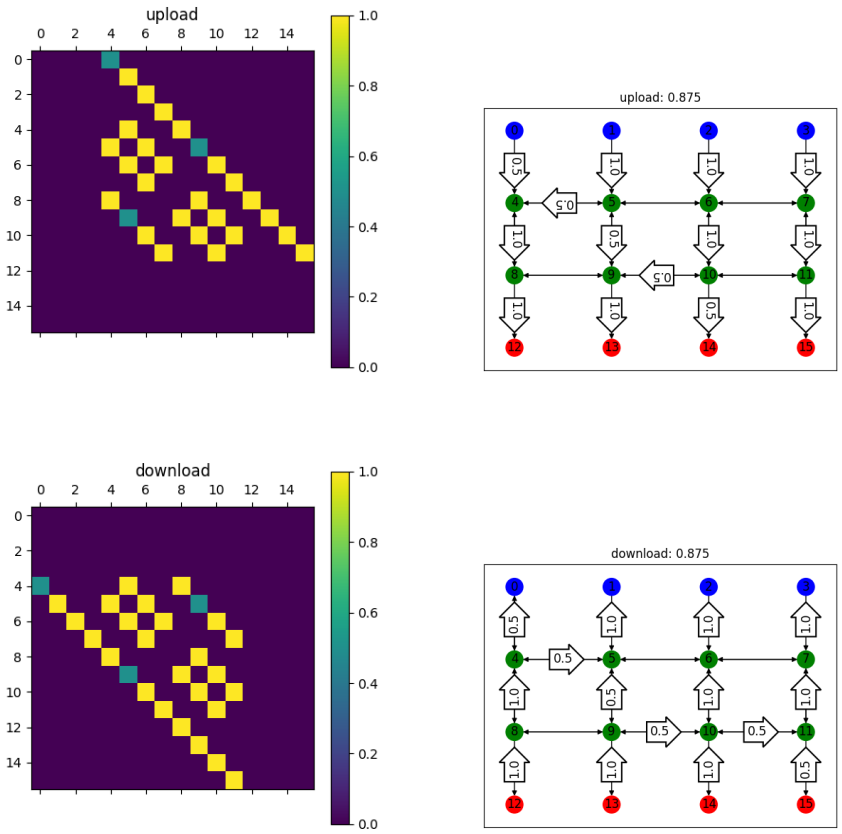


Slika 4: Prikaz hitrosti prenosa podatkov v popolnem omrežju.

Program je sam prišel do enake rešitve kot mi, to mi daje zaupanje da program deluje dobro. Poglejmo si še nekaj težjih primerov, kjer vse povezave ne delujejo, a si vseno še znamo predstavljati optimalno rešitev.

3.2 Realno omrežje

Relano omrežje lahko poskusimo simulirati z "pokvarjenimi žicami", torej povezave med nekaterimi vozlišči zmanjašmo na poljubno vrednost, a hkrati vseno poskrbimo da zmanjšamo malo število povezav, da bomo problem lahko rešili v glavi, ter tako potrdili ali program deluje dobro. Zmanjšamo lahko naprimer povezavo med vozliščema 0 in 4, ter vozliščema 5 in 9 (slika 5).



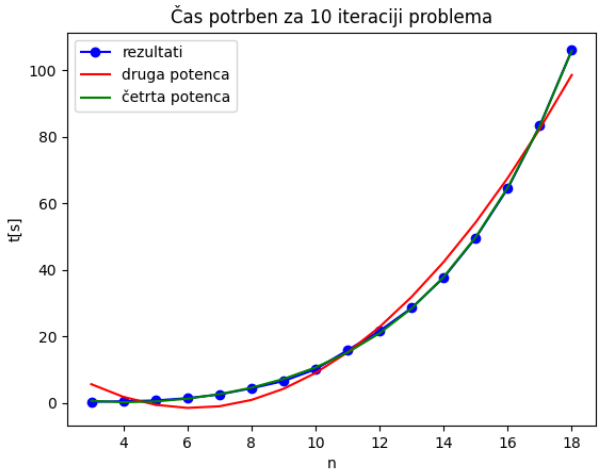
Slika 5: Prikaz hitrosti prenosa podatkov v omrežju z dvema slabšima povezavama.

Na sliki 5 opazimo nekaj zanmivih stvari:

- če so povezave enake v obe smeri (od i-tega do j-tega vozlišča in od j-tega do i-tega vozlišča) dobimo enako največjo hitrost za upload in download
- pot podatkov ni nujno ista za obe smeri
- pot podatkov ni enaka kot bi jo predlagal človek (za primer lahko pogledamo upload. V vozlišče 10 priteka hitrost 1, enostavna rešitev bi bila, da se vsa ta hitrost steka v strežniki 14, program pa je določil, da se v strežniki steka samo 1/2, druga polovica pa gre v vozlišče 9 in nato naprej v strežnik 13). Končna rešitev (makismalna hitrost) je sicer enaka, kot bi jo izračunal človek, zato program še vedno deluje.

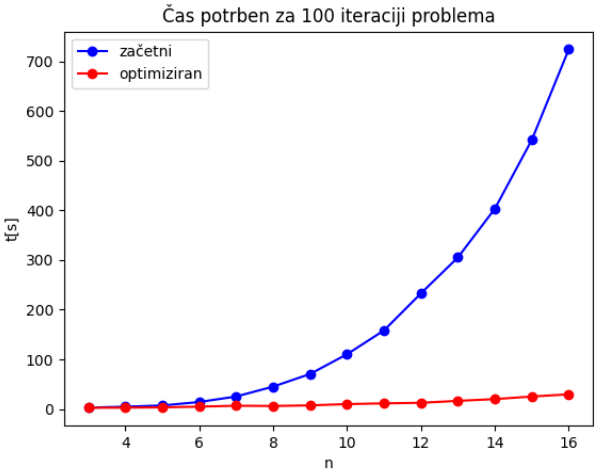
3.3 Časovna zahtevnost algoritma

Kot pri večini algoritmov nas tudi tukaj zanima časovna zahtevnost. Število vozlišč v sistemu narašča kot n^2 število povezav pa kot n^4, saj je vsako vozlišče povezano z vsakim. Večina teh povezav je sicer v programu omejena na minimalno vrednost 0 in maksimalno vrednost 0, kar pomeni, da jih program ne potrebuje optimizirati. Poglejmo si s katerim od teh redov narašča časovna zahtevnost programa.



Slika 6: Časovna zahtevnost osnovnega algoritma. Fit z drugo potenco ($f(x) = ax^2 + bx + c$) se ne prilega časovni zahtevnosti. Medtem ko se fit s četro potenco ($f(x) = ax^4 + bx^3 + cx^2 + dx + e$) skoraj popolnoma prilega rezultatom. ($a = 0.00247$, $b = -0.05853$, $c = 0.793$, $d = -4.24$, $e = 7.5339$)

Časovna zahtevnost narašča s četrto potenco (številom povezav), kljub temu, da je večina povezav omejenih na 0. Program lahko verjetno optimiziramo z odstarnitev vseh povezav ki so ža na začetku nastavljene na 0 (povezave med nesosednjimi ogljišči). V tem primeru bo število povezav naraščalo približno kot $4n^2$, saj ima vsako ogljišče maksimalno 4 sosede. Poglejmo si primerjavo časovnih zahtevnosti začetnega in optimiziranega programa.



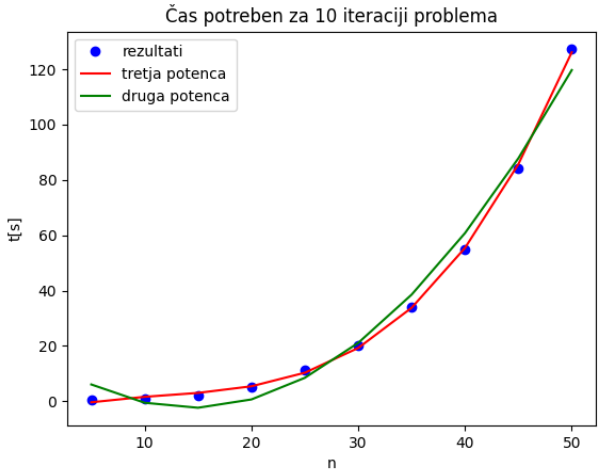
Slika 7: Primerjava časovnih zahtevnosti obeh algoritmov.

Kot vidimo deluje drugi algoritem precej hitreje za večje sisteme. Poglejmo si še njegovo časovno zahtevnost.

Kot vidimo na sliki 8 časovna zahtevnost optimiziranega algoritma narašča z n^3, točnega razloga za to sicer nimam, ampak vsaj program deluje precej hitrej kot prej in se lahko sedaj lotim analize statisične porazdelitve zanmivih količin. Saj bom za te del moral za dosti veliki vzorec precej veliko poganjati.

3.4 Statistična porazdelitev količin

V tem delu naloge sem maksimalne hitrosti povezav nastavil na naključno vrednost med 0 in 1. Program sem nato za vsak n pognal velikokrat in izračunal povprečje določenih količin. Najprej me je zanimala povprečna hitrost prenosa v odvisnsoti od velikosti sistema n. Za velikost $n = 2$ lahko to povprečno maksimalno hitrost ocenimo, saj imamo v tem primeru direktno povezavo med strežnikom in odjemalcem (naj spomnim, da strežniki niso povezani med sabo, prav tako tudi računalniki, torej imamo v tem primeru



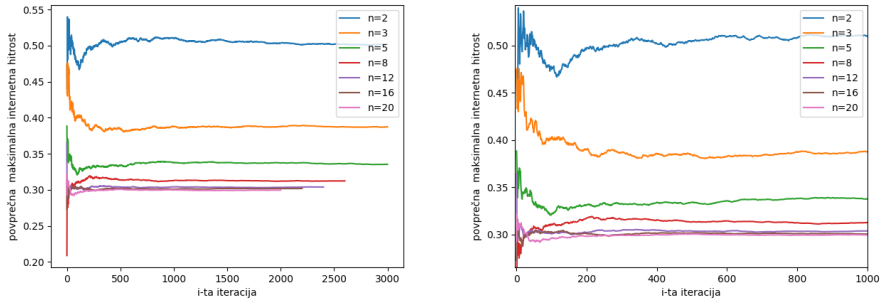
Slika 8: Časovna zahtevnost optimiziranega algoritma. Fit z drugo potenco ($f(x) = ax^2 + bx + c$) se ne prilega časovni zahtevnosti. Medtem ko se fit s tretjo potenco ($f(x) = ax^3 + bx^2 + cx + d$) skoraj popolnoma prilega rezultatom. ($a = 0.0020$, $b = -0.0725$, $c = 1.1246$, $d = -4.4212$

dva popolnoma ločena sistema). Ta povezava je v vsaki iteraciji naključna vrednost med 0 in 1, torej pričakujemo, da bo njeno povprečje čez veliko število iteracij 1/2. Za večje sisteme točno vrednost težje napovemo, a pričakujem da bo z večanjem n padala. Poglejmo si najprej koliko iteracij porgram potrebuje da se stabilizira za določeno veliksto sistema (slika 9).

Kot prvo opazimo, da je povprečna maksimalna hitrost za najmanjši sistem ($n = 2$) 1/2 kot sem napvoedal, ter da za večje sisteme ta vrednost pada. Prav tako opazimo, da večji sistemi potrebujejo manj iteracij da pridejo do "konstantnega" rezultata (program se je predčasno ustavil, če je bilo nihanje rezultata v zadnjih 100 iteracijah manjše kot 10^{-4}. Na desni sliki 9 vidimo, da potrebujemo več kot 1500 iteracij, da lahko ločimo med večjimi sistemi (na podlagi končnega rezultata).

Naslednja stvar, ki sem jo želel raziskati je hitrost posameznega računalnika v sistemu. Zanimalo me je ali je ta konstantna ali ima kateri od računalnikov v povprečju večji internetni prenos podatkov (slika 11).

Kot vidimo sta v večini primerov najpočasnejša "robna" računlnika, drugih vzorcev pa ne opazim. Prav tako nisem prepričan, da je smiselno da bi jih iskal, saj ima večina problemov več kot eno rešitev, program pa vedno vrne prvo rešitev, ki jo najde. Zanimiv probelm bi lahko bila tudi maksimizacija minimalne internetne hitrosti za vsak posamezen računalnik, to bi nas



Slika 9: Spreminjnanje povprečne maksimalne hitrosti za različne velikosti sistema (levo). Ter povečano "zanimivo" območje, kjer prihaja do večjih nihanj.

verjetno privedlo do bolj konstantne hitrosti med uporabniki.

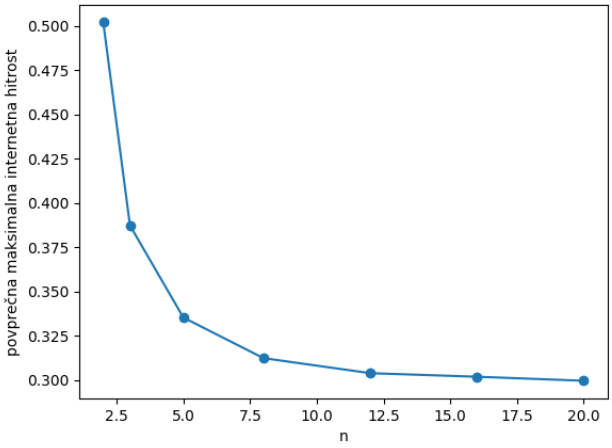
Zanima nas tudi kako na končno internetno hitrost v posameznem sistemu vpliva hitrost določenih povezav. Preveril bom koreleacijo med končno hitrost in naslednjimi lastnostmi sistema:

- Povprečno maksimalno dovoljeno hitrost vodoravnih povezav v sistemu. V tem primeru ne pričakujem močne korelacije, saj lahko dobimo maksimalno hitrost, kljub temu, da so izključene vse vodoravne povezave (če so vse navpične povezave maksimalne).

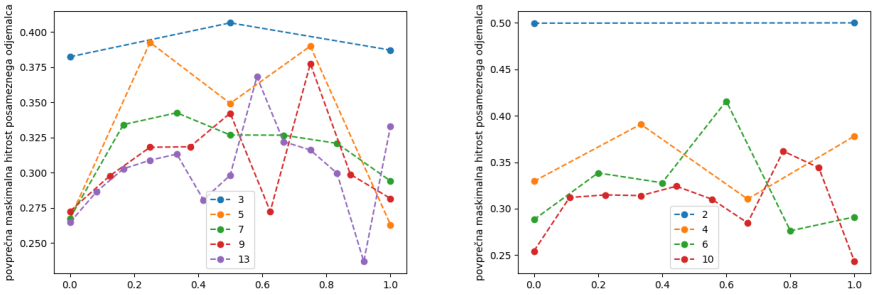
- Povprečno maksimalno dovoljeno hitrost navpičnih povezav v sistemu. V tem primeru pričakujemo zelo veliko korelacijo, saj navpične povezave povezujejo strežnike in odjemalce.

- Povprečno izkoriščeno hitrost vodoravnih povezav v sistemu. Podbno kot v prvem primeru tu ne pričakujem velike koralcije, a vseno večjo kot v prvem primeru.

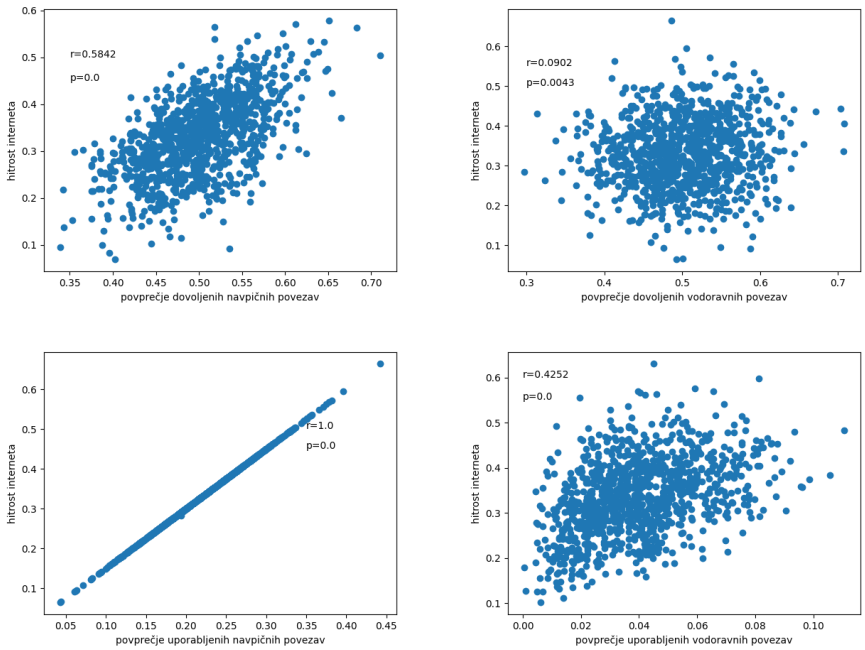
- Povprečno izkoriščeno hitrost navpičnih povezav v sistemu. V tem priemru pa pričakujemo največjo korelacijo od vseh omenjehni primerov.



Slika 10: Povprečna maksimalna hitrost v odvinosti od n.



Slika 11: Povrečna hitrost posamezenega računalnika v sistemu za liho (levo) in sodo (desno) število odjemalcev .



Slika 12: Prikaz končne hitrosti v odvinosti od zgoraj naštetih lastnosti ter Pearsanovega korelacijskega koeficijenta (r) in p-vrednosti.