ANALIZA REŠEVANJA METODE K VODITELJEV Z UPORABO IMPLEMENTIRANEGA ALGORITMA

Valentina Gnamuš (mentor: prof. dr. Sergio Cabello)

18. 2. 2017

Problem metode k voditeljev (z manhattansko razdaljo)

Opis problema:

Metoda voditeljev (k-median) je eden najpogosteje proučevanih principov razvrščanja v skupine. Pri metodah razvrščanja v skupine želimo objekte neke množice razvrstiti v skupine tako, da si bodo objekti znotraj posameznih skupin čimbolj podobni med seboj (glede na določeno lastnost) in objekti različnih skupin čimbolj različni med seboj.

Metoda voditeljev (k-median) sodi med nehierarhične metode razvrščanja, kar pomeni, da je treba vnaprej podati število skupin iskane razvrstitve (v našem primeru je to število t.i. k). Je iteracijska metoda, ki je zelo popularna, saj zmore v skupine razvrščati večje število enot.

V problemu imamo podano množico točk S in želeno število voditeljev k. Izmed točk, ki so v množici S želimo izbrati k voditeljev tako, da bo vsota razdalj od vsake točke do najbližjega centra najmanjša.

O. Kariv in S. L. Hakimi sta dokazala, da je ta problem NP-težek, kar pomeni, da zanj ni rešitve, ki bi se izvedla v polinomskem času. Najpreprostejši algoritem, ki bi rešil problem, bi pregledal vsako možno množico voditeljev (velikosti k), izračunal vsote razdalj točk do teh centrov in na koncu vrnil najboljšo možno skupino centrov z najmanjšo vsoto razdalj. Vendar bi tak algoritem do rešitve prišel šele v $\mathcal{O}(\binom{n}{k}nk)$ času, pri čemer je k število voditeljev in n velikost vzorca. Tako so se akademske raziskave na tem področju skozi čas osredotočile predvsem na razvijanje dobrih aproksimativnih algoritmov.

Zapis problema kot celoštevilski linearni program:

Naj bo y_i binarna slučajna spremenljivka, ki za vsak element iz množice točk S vrne 1, če je dani element izbran za voditelja in 0, če ni. Naj bo $x_{i,j}$ binarna slučajna spremenljivka, ki vrne vrednost 1, če ima element j iz množice S element i za voditelja, in 0 sicer. Problem formuliramo kot celoštevilski linearni program na sledeč način:

$$\min \sum_{i,j \in S} d_{ij}\, x_{ij}$$

$$p.p. : \forall j \in S : \sum_{i \in S} x_{ij} = 1$$

$$\forall i, j \in S : x_{ij} \leq y_i$$

$$\sum_{i \in S} y_i \leq k$$

$$\forall i, j \in S : x_{ij} \in \{0, 1\}\,,\; y_i \in \{0, 1\}$$

Pri tem prva omejitev pove, da ima vsaka točka natančno enega voditelja, druga omejitev pove, da je spremenljivka x_{ij}lahko 1 le, če je točka i voditelj in tretja omejitev pove, da je vseh voditeljev največ k. Naša ciljna funkcija pa najde rešitev z najmanjšo vsoto razdalij (razdalje so shranjene v vrednostih d_{ij}).

Celoštevilski linearni program smo zapisali v sage-u, rešitev le-tega nam vrne funkcija CLP. Koda je vidna tudi na GitHubu.

Reševanje s požrešno metodo

Opis metode:

Požrešni algoritem je eden izmed hevrističnih algoritmov, kar pomeni, da problema ne reši optimalno, ampak na vsakem koraku izbere lokalno optimalno rešitev. Za metodo k voditeljev obstaja več različnih požrešnih algoritmov, eden izmed njih je tako imenovani "napredujoči požrešni algoritem", saj začnemo s praznim seznamom voditeljev, ki ga nato na vsakem koraku dopolnemo s točko, ki nam skupaj z že ostalimi izbranimi voditelji vrne najmanjšo vsoto razdalj, dokler ta seznam ni velikosti k. Tak način reševanja problema nam tako vrne neko zadovoljivo rešitev, ki pa v splošnem ni optimalna. Požrešni algoritmi namreč nikoli ne zagotavljajo optimalne rešitve, kljub temu pa z njimi lahko pridemo do solidnih rezultatov. V kasnejših poskusih si bomo pogledali tudi, koliko se rešitve, ki jih dobimo s požrešno metodo razlikujejo od rešitev, ki jih dobimo na druge načine.

Zapis psevdokode za požrešno metodo:

Z napredujočim požrešnim algoritmom problem rešujemo na sledeči način:

- nastavimo množico voditeljev: $V = \{\}$
- dokler $|V| < k$ ponavljamo: f=arg min$_i(cena(V \cup \{i\}))$
- $V = V \cup \{f\}$

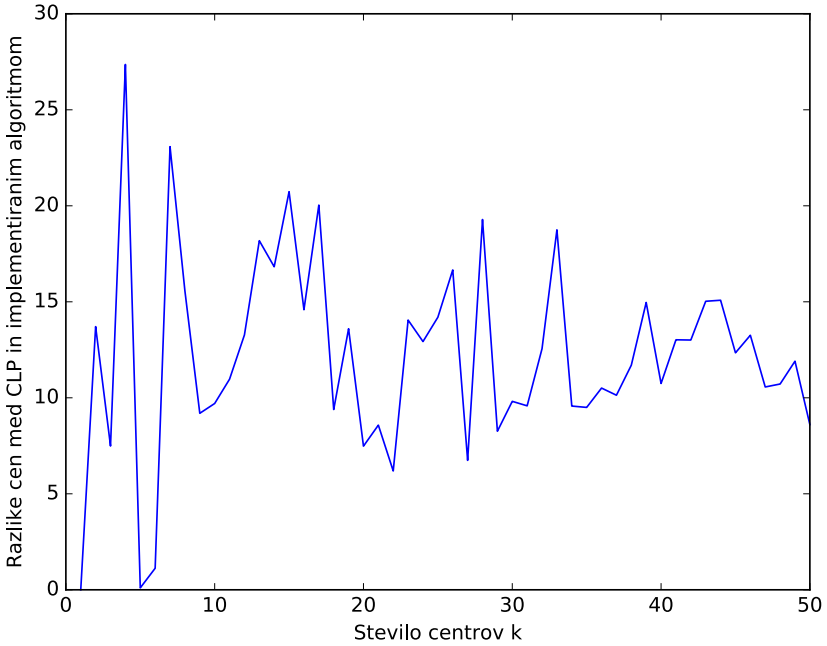
Takšen algoritem naj bi problem v praksi rešil dokaj hitro, vendar pa je število korakov najmanj $\mathcal{O}(n)$. Algoritem je za problem zapisan v Pythonu, koda pa je objavljena tudi na GitHubu.

Reševanje na druge načine

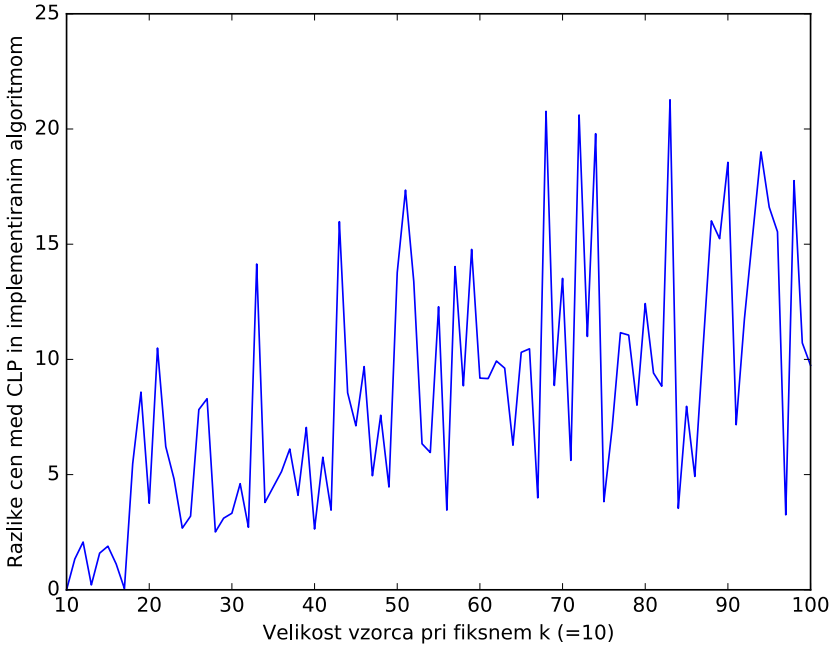
Problem smo rešili še na en način in sicer tako, da smo podali naključni izbor k voditeljev, ki smo ga nato spreminjali dokler se je vsota razdalj še spreminjala. Algoritem smo zapisali v Pythonu, kodo, iz katere smo izhajali, pa smo si sposodili iz knjižnice `pyclustering`, avtor kode pa je Andrei Novikov. Tako so razred kmedioids in ostale funkcije, ki jih je bilo potrebno definirati za delovanje algoritma, shranjene v datoteki `optimal.py`, sam algoritem za uporabo pa zapisan v datoteki `algoritem.py`. Vse te datoteke so objavljene na GitHubu.Tudi ta način bomo v nadaljevanju primerjali z ostalima dvema, in sicer bomo primerjali učinkovitost posameznih metod.

Rezultati in ugotovitve

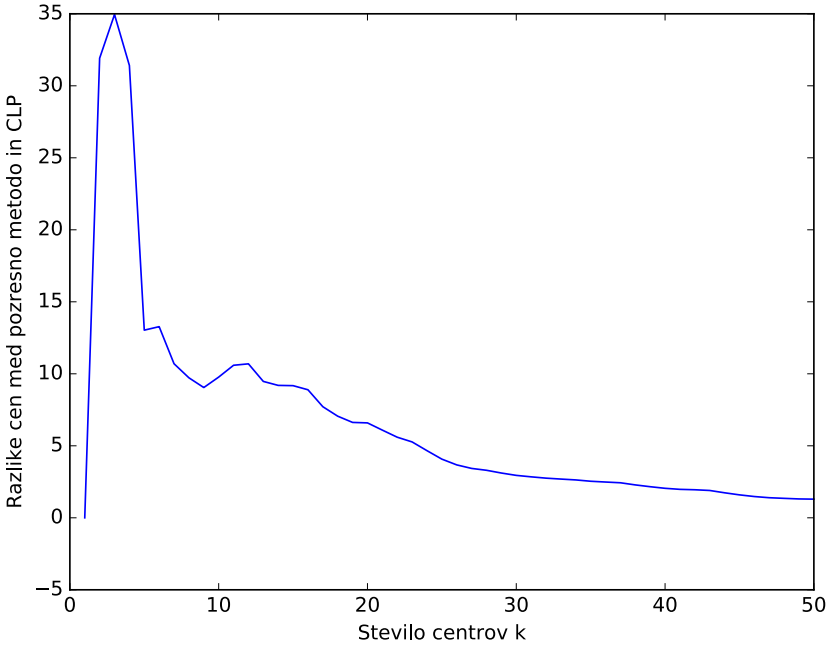
Metode reševanja smo kar nekajkrat pognali in dobili zanimive rešitve. Primerjali smo cene med posameznimi algoritmi in gledali, kako se rešitve razlikujejo, če spreminjamo število voditeljev, in kako, če spreminjamo velikost vzorca pri fiksnem k. Prvi graf, ki smo ga narisaki, je izpisal rezultate pri vzorcu velikosti 100, pri čemer smo 100 dvodimenzionalnih točk generirali naključno z uporabo funkcije random v pythonu. Število k smo tu spreminjali na vsakem koraku, od 1 do 50. Nato smo narisali še en graf, ki pa je pokazal, kako se rezultati spreminjali pri fiksnem k (izbrali smo si $k = 10$), če velikost vzorca spreminjamo (velikost vzorca smo povečevali od 10 do 100). Grafi so pokazali sledeče rezultate.



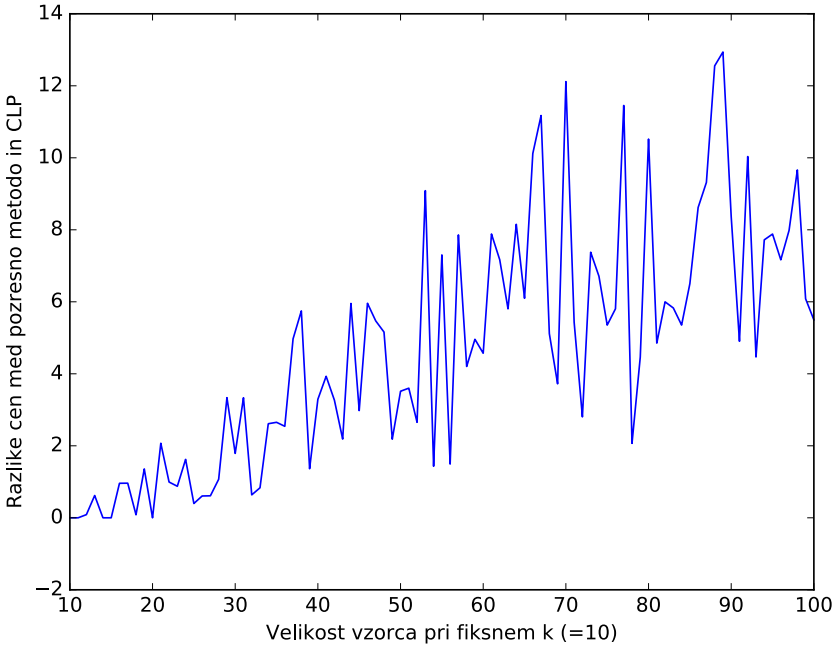
Sprva smo primerjali implementiran algoritem iz modula `pyclustering` in celoštevilski linearni program. Algoritma smo pognali 50-krat pri fiksnem vzorcu velikosti 100, pri čemer smo na vsakem koraku število voditeljev povečali za 1. Kot vidimo, je implementiran algoritem na vsakem koraku vrnil slabšo rešitev kot celoštevilski linearni program, razlika med dvema je vedno pozitivna. Razlika v rešitvah tudi precej oscilira, vendar pa se, ko povečujemo število voditeljev, oscilacija zmanjšuje.



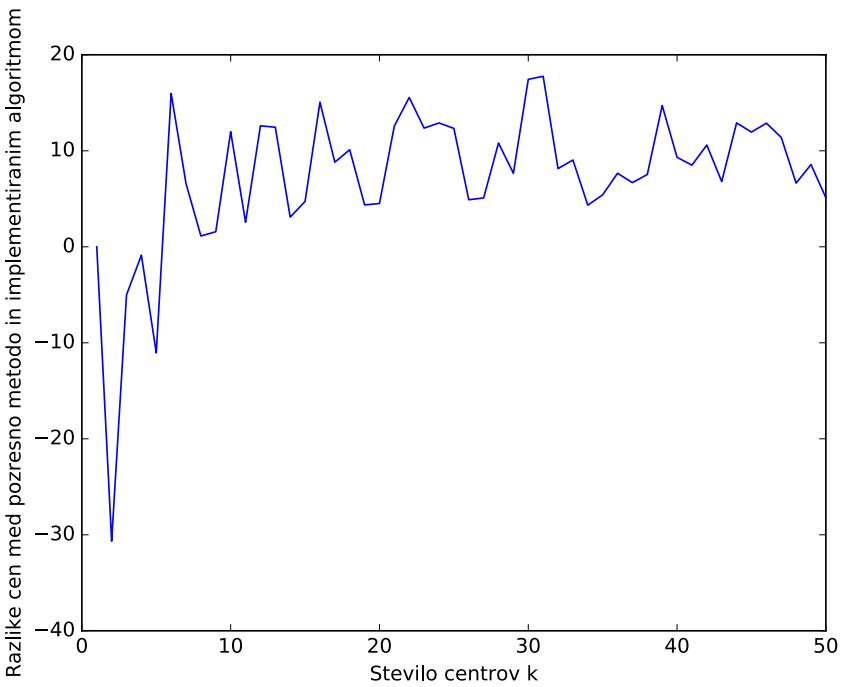
Implementiran algoritem iz `pyclustering` ter celoštevilski linearni program smo primerjali še v primeru, ko k fiksiramo in spreminjamo velikost vzorca. Vidimo lahko, da razlika v rešitvah oscilira bolj, ko se velikost vzorca oddaljuje od števila voditeljev, kar je smiselno, če rezultat primerjamo s prejšnjim grafom.



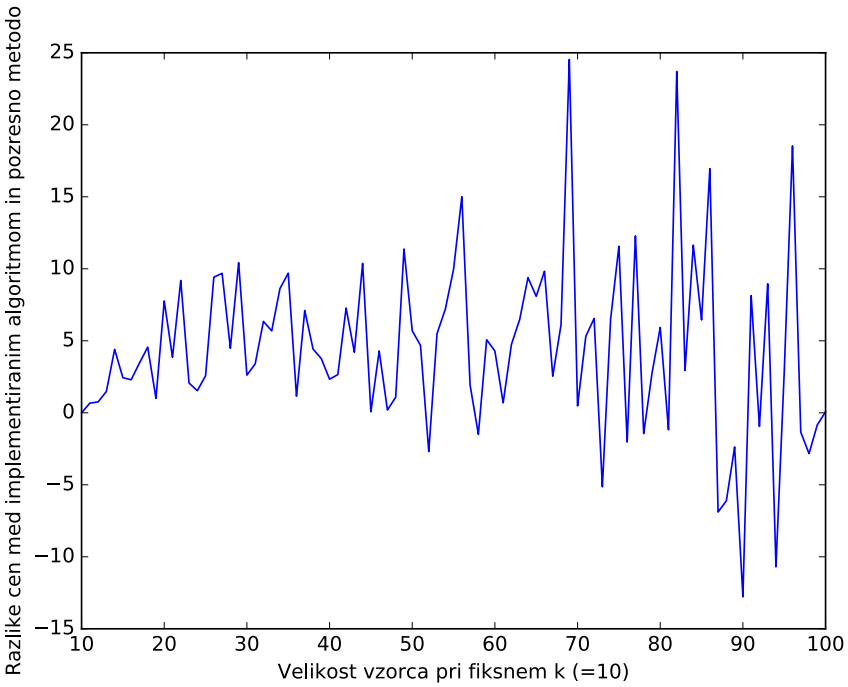
Naslednja dva algoritma, ki smo ju primerjali, sta bila napredujoči požrešni algoritem ter celoštevilski linearni program. Vidimo lahko, da tudi požrešna metoda vedno vrne slabši rezultat kot osnovni CLP, kljub temu pa se njegov približek v rešitvi optimalni rešitvi približuje eksponentno hitro, ko večamo število voditeljev k. Ko smo primerjali rešitve pri vedno večjem n, smo dobili naslednji graf:



Vidimo lahko, da ko povečujemo velikost vzorca, je delež voditeljev v vzorcu vedno manjši, razlika v končni vsoti pa se postopno povečuje in vedno bolj oscilira. Torej požrešna metoda vrne boljše rezultate, če je število voditeljev večje.



Nazadnje smo primerjali še dve različni metodi reševanja problema, pri čemer nobena od teh dveh ne vrne optimalne rešitve. V grafu gledamo razlike v rešitvah, ki jih vrne požrešna metoda, in rešitvah, ki jih vrne algoritem iz modula `pyclustering` (tako da od rešitve implementiranega algoritma odštejemo rešitev požrešnega algoritma). Vidimo lahko, da je za majhne k uporabnejši algoritem iz modula `pyclustering`, saj da boljše rešitve. Ko pa pridemo do $k = 10$ pa imamo naprej same pozitivne vrednosti v grafu, kar pomeni, da boljšo rešitev da požrešna metoda.

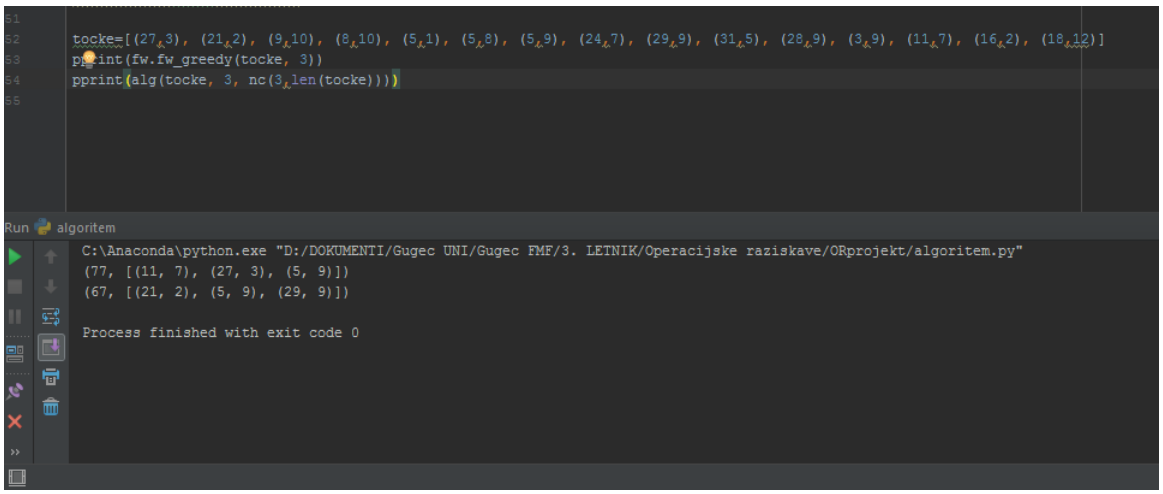


Tudi ko smo oba algoritma 91-krat pognali, pri čemer smo na vsakem koraku vzorec povečali za 1, smo dobili zanimiv rezultat. Torej pri fiksnem k v splošnem boljšo rešitev dobimo s požrešno metodo, v kolikor sta velikost vzorca in število voditeljev števili, ki sta si po velikosti dokaj blizu. Ko pa se z velikostjo vzorca oddaljujemo od števila voditeljev, ki jih iz vzorca izberemo, pa se rešitvi posameznih alternativnih metod začneta vedno bolj razhajati, zato ne moremo z gotovostjo trditi, da je eden izmed teh aproksimativnih algoritmov boljši.

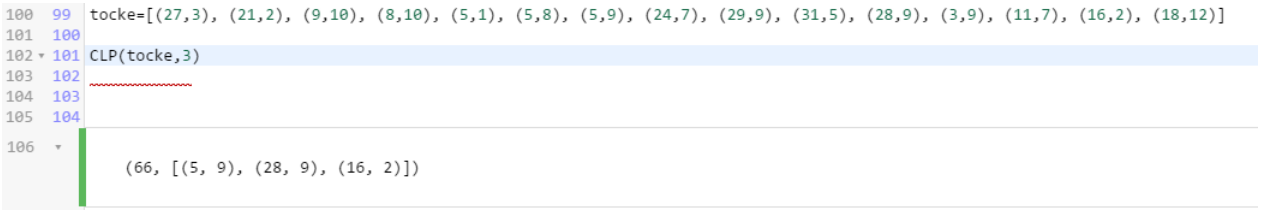
Rešitev konkretno zapisanega problema

Zanimala nas je še rešitev konkretno zapisanega problema, zato smo generirali še majhen problem, vzorec iz 15 točk, in iskali 3 centre tako, da bi bila vsota razdalj do najbližjih centrov najmanjša.

Aproksimativna algoritma sta vrnila sledeči rešitvi:



Medtem, ko je bil rezultat optimalnega programa:



Vidimo, da je bil v tem primeru algoritem iz modula `pyclustering` učinkovitejši, saj je bil k petkrat manjši od vzorca. Drugi algoritem je tudi za center izbral $\frac{2}{3}$ istih točk kot optimalen algoritem, medtem ko je požrešna metoda pravilno določila le 1 center.

Sklep

Torej, kot smo videli, je problem izbire k voditeljev poznan problem, o katerem se vztrajno piše in išče novih, hitrejših poti do rešitev. Metoda k voditeljev je zelo uporaben problem v praksi, obravnava se na številnih strokovnih področjih.

Optimalna rešitev tega problema nam sicer da zadovoljiv rezultat, vendar pa je pot do tega rezultata ob veliki količini podatkov izredno dolga. Zato že obstaja vrsta aproksimativnih algoritmov, s pomočjo katerih lahko pridemo do zadovoljivih rezultatov. Eden izmed teh je napredujoči požrešni algoritem,

drugi je naprimer algoritem, ki naključne centre spreminja, dokler ne pride do zadovoljivih rešitev. Oba smo v projekni nalogi analizirali in prišli do zanimivih ugotovitev.

References

[1] K-Median Algorithms: Theory in Practice. 2015. [internet].[citirano 17.2.2017]. Dostopno na naslovu: D. Dohan, S. Karp, B. Matejek

[2] K-medians clustering. 2016. [internet].[citirano 17.2.2017]. Dostopno na naslovu: wikipedia

[3] Lecture 2 - The k-median clustering problem. 2013. [internet].[citirano 17.2.2017]. Dostopno na naslovu: Ucsd Dasgupta Lectures

[4] Python Help. 2001-2017. [internet].[citirano 8.2.2017]. Dostopno na naslovu: Python Help