Univerza v Ljubljani
Fakulteta za matematiko in fiziko

Finančni praktikum

Največje neodvisne množice z lokalnim iskanjem

Avtorja:
Jaka Mrak
Žiga Gartner

Mentorja:
prof. dr. Sergio Cabello
doc. dr. Janoš Vidali

Ljubljana, 10. januar 2022

Kazalo

Slike

1 Navodilo

Naloga je iskanje največje neodvisne množice v grafu $G = (V, E)$ s pomočjo celoštevilskega linearnega programiranja. Velike neodvisne množice v grafu lahko poiščemo s pomočjo metode lokalnega iskanja. Začnemo s poljubno neodvisno množico $U \subseteq V$, kjer k vozlišč nadomestimo s $k + 1$ vozlišči tako, da ohranjamo neodvisnost množice U. Konstanta k je dana na začetku. Primerjali bomo metodi lokalnega iskanja in optimalne rešitve ter primerjali njune rešitve za nekatere preproste grafe.

2 Opis problema

Definicija 1. *Naj bo $G = (V, E)$ graf.* ***Neodvisna množica*** *U, v grafu G, je taka podmnožica množice vozlišč V, kjer poljubni dve vozlišči iz množice U nista sosednji.* ***Maksimalna neodvisna množica*** *v grafu G pa je taka neodvisna množica, kjer ne obstaja vozlišče $v \in V$ in $v \notin U$, ki bi ga lahko dodali množici U in pri tem ohranili neodvisnost množice U. Torej je neodvisna množica U največja taka, če velja ena od naslednjih dveh lastnosti:*

1. *$v \in U$*

2. *$S(v) \cap U \neq \emptyset$, kjer je $S(v)$ množica sosedov v.*

Največja neodvisna množica *je neodvisna množica, največje možne velikosti, za dan graf G. Velikosti največje neodvisne množice, za graf G, pa pravimo* ***neodvisnostno število*** *in pogosto označimo $\alpha(G)$.*

Definicija 2. *Celoštevilski linearni program v standardni obliki je dan z matriko $A \in \mathbb{R}^{m \times n}$, vektorjem $b \in \mathbb{R}^m$ in vektorjem $c \in \mathbb{R}^n$. Iščemo*

$$max < c, x >,$$

da bodo zadoščeni pogoji

$$Ax \leqslant b, x \geqslant 0,$$

kjer je $x \in \mathbb{Z}^n$.

Posledica 1. *Problem največje neodvisne množice v grafu $G = (V, E)$ lahko s celoštevilskim linearnim programiranjem modeliramo na sledeč način:*

$$max \sum_{v \in V} x_v,$$

da velja:

$$x_v + x_w \leqslant 1 \; za \; \forall vw \in E,$$

$$x_v \in \{0, 1\}$$

$$x_u = \begin{cases} 1, & za \; u \in U \\ 0, & za \; u \notin U \end{cases} \quad , U \text{ neodvisna množica v grafu } G.$$

Največjo neodvisno množico v množici vseh neodvisnih podmnožic grafa $G = (V, E)$ bomo iskali s pomočjo celoštevilskega lineranega programiranja in lokalnega iskanja. **Lokalno iskanje** temelji na izbiri začetne neodvisne podmnožiče vozlišč $U \subset V$ v kateri k vozlišč zamenjamo s $k + 1$ vozlišči in pri tem ohranjamo neodvisnost množice U.

3 Opis dela

V nadaljevanju bomo največje/maksimalne neodvisne množice iskali s pomočjo CLP v Sage in s pomočjo implementiranega algoritma $nakljucni\_MIS(G)$, kasneje izboljšanega z lokalnim iskanjem. Algoritme bomo izvajali na grafih, generiranih s pomočjo Erdős-Rényijevega $G(n, p)$ modela. Primerjali bomo maksimalne (ali pa največje) neodvisne množice, ki jih algoritmi poiščejo, časovno zahtevnost algoritmov, kasneje pa še, kako vpliva spreminjanje števila vozlišč in verjetnosti, v modelu, na velikost maksimalne (ali pa največje) neodvisne množice in na časovno zahtevnost algoritmov.

3.1 Generiranje podatkov

Kot omenjeno, bomo grafe generirali s pomočjo Erdős-Rényijevega $G(n, p)$ modela, kjer podamo algoritmu parameter n za število vozlišč in p za verjetnost povezave med poljubnima vozliščema v grafu.

Definicija 3. *Erdős-Rényijev model $G(n, p)$ generira graf z n naključno povezanimi vozlišči. Vsaka povezava je v graf vključena neodvisno, z verjetnostjo p.*

Generirali bomo grafe s konstantima vrednostma (n, p) nato z naraščajočim n in konstantnim p in nazadnje še s konstantim n in naraščajočim p. Za generacijo podatkov v Pythonu bomo uporabili knjižnico $NetworkX$. Da bomo lahko grafe uporabljali še v okolju $Sage$, bomo le-te, s funkcijo $NetworkX.to\_dict\_of\_lists(graf)$, pretvorili v slovarje seznamov. Grafe, na katerih bomo izvajali algoritme, bomo shranili v $JSON$ datoteke, da imamo evidenco na kakšnih grafih smo izvajali algoritme.

3.2 Algoritmi

Za iskanje maksimalne neodvisne množice grafa $G = (V, E)$ smo implementirali algoritem $nakljucni\_MIS(G)$, čigar rešitev bomo poizkušali izboljšati še z algoritmom $lokalno\_iskanje(G, I)$. Oba algoritma bosta kot argument sprejela graf G, algoritem $lokalno\_iskanje(G, I)$ pa še neko maksimalno neodvisno množico I grafa G. Poleg tega sva največje neodvisne množice iskala še s CLP prilagojenim za naš problem.

3.2.1 Algoritem *CLP*

Algoritem 1 $CLP(G)$

1: $max \sum_{v \in V} x_v$

2: da velja $x_v + x_w \leqslant 1$ za $\forall vw \in E$

3: $x_u = \begin{cases} 1, & za\ u \in U \\ 0, & za\ u \notin U \end{cases}$, U neodvisna množica v grafu G.

3.2.2 Algoritem $nakljucni\_MIS(G)$

Algoritem 2 $nakljucni\_MIS(G)$

$I = \emptyset$
$\forall v \in V$ dobi vrednost $P(v) \in permutacija(V)$
if $P(v) < P(w)$ *za* $\forall w \in sosedi(v)$ **then**
    $I = I \cup v$
$V' = V \backslash (I \cup sosedi(I))$.
$E' = E \backslash povezave(I)$.
return $I \cup MIS(G' = (V', E'))$

3.2.3 Algoritem $lokalno\_iskanje(G, I)$

Najprej definirajmo $\tau(v)$, v algoritmu imenovan $tightness(v)$.

Definicija 4. *Naj bo* $G = (V, E)$ *in* $v \in S \subset G$. *Potem je:*

$$\tau(v) = |W|, \; kjer \; je \; W = \{w | w \in (sosedi(v) \cap V \backslash S)\}.$$

Algoritem $lokalno\_iskanje(G, I)$ sprejme graf G in neko maksimalno neodvisno množico I v grafu G. Nato vsakemu vozlišču $v \in I$ priredi mnozico $L(v) = \{w \in sosedi(v) | \tau(w) = 1\}$. Nato v vseh možnih parih $(x, y) \in L(v) \times L(v)$, ki zadoščajo $x \neq y$, poišče prvega, da velja $y \notin sosedi(x)$, ko najde prvi tak par iz množice I odstrani vozlišče v, in množici I doda vozlišči x in y. Algoritem vrne maksimalno neodvisno množico z lokalnim iskanjem takrat, ko ne obstaja več vozlišče v neodvisni množici, ki bi ga lahko zamenjali z dvema.

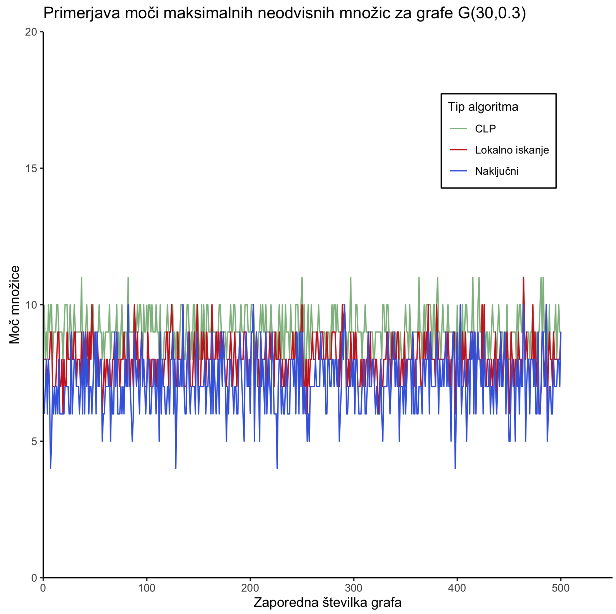
```
import networkx as nx

def tightness(graf, mnozica, v):
    t = 0
    for w in graf.neighbors(v):
        if w in mnozica:
            t += 1
        else:
            pass
    return t

    def lokalno_iskanje(G, I):
    I = list(I) # naredimo kopijo seznama I, oziroma ga pretvorimo, ce ni v obliki seznama
    J = [] # seznam vozlisc, ki niso bila nadomescena
    while I: # ponavljamo, dokler seznam ni prazen
        v = I.pop(0) # poberemo prvo vozlisce iz I
        L = [w for w in G.neighbors(v) if tightness(G, I+J, w) == 0]
        for i, v1 in enumerate(L):
            for w1 in L[i+1:]: # kopiramo iteratorja - da pregledamo vsak par enkrat
                if w1 not in G.neighbors(v1):
                    I += [v1, w1] # dodamo vozlisci v I
                    break
            else: # notranja zanka je prisla do konca
                continue # poskusimo z drugim v1
            break # notranjo zanko smo prekinili, prekinimo se zunanjo
        else: # zunanja zanka je prisla do konca
            J.append(v) # vozlisca v nismo uspeli nadomestiti, pustimo ga v izhodu
    return J
```

3.3 Analiza rezultatov

3.3.1 Analiza algoritmov na grafih $G(30, 0.3)$

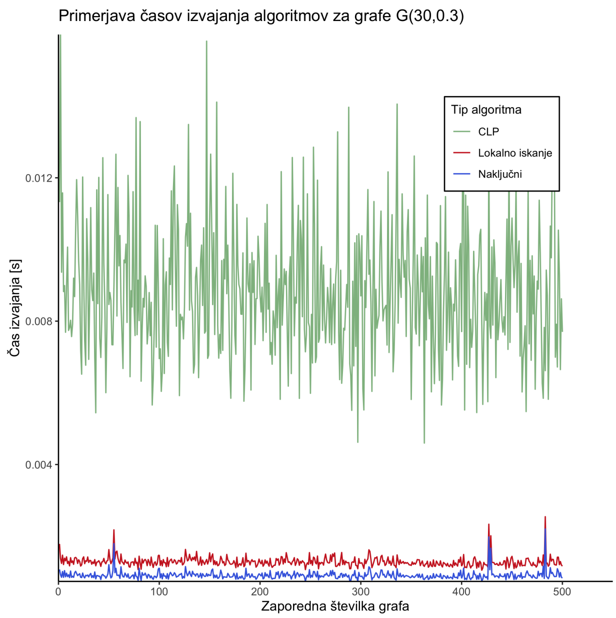
Najprej bomo primerjali algoritme na Erdős-Rényijevih $G(30, 0.3)$ grafih. Na tak način sva generirala 500 grafov in na njih izvedla algoritma $CLP(G)$, $nakljucni\_MIS(G)$, za vsak graf pa sva slednjega poskušala izboljšati še z algoritmom $lokalno\_iskanje(G, I)$, ki poleg grafa G sprejme še $I = nakljucni\_MIS(G)$.

Opazimo, da neodvisne množice največjih moči najde CLP. Variacije moči neodvisnih množic, ki jih lahko vidimo na spodnjem grafu, so rezultat naključnosti pri generiranju $G(30, 0.3)$ grafov. Od slučaja je namreč odvisno, kako bodo postavljene povezave v grafu G.



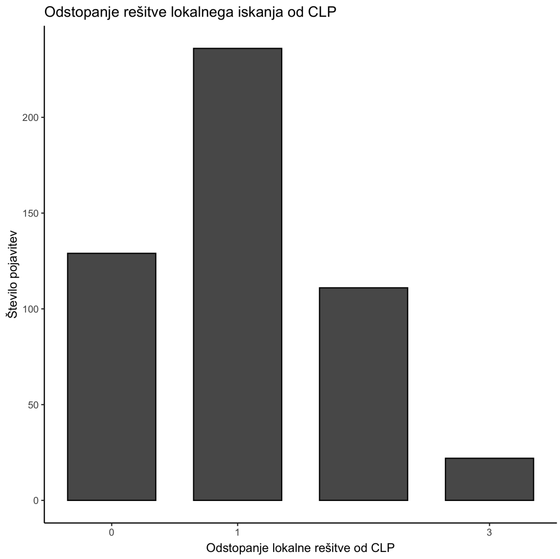
Slika 1: Moči neodvisnih množic za grafe $G(30, 0.3)$

Kljub temu, da je, za generirane grafe, CLP najpogosteje vrnil najboljšo rešitev, lahko v naslednjem grafu vidimo njegovo slabost. CLP je namreč občutno počasnejši od preostalih dveh algoritmov.



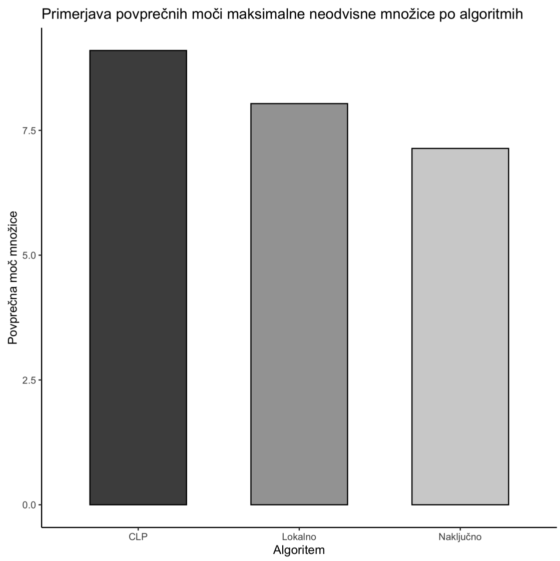
Slika 2: Časovne zahtevnosti algoritmov za $G(30, 0.3)$

Poglejmo si sedaj še odstopanja rešitev *lokalnega_iskanja* od rešitev dobljenih s CLP. Hitro pazimo, da je algoritem *lokalno_iskanje* zelo učinkovit pri iskanju maksimalne neodvisne množice, saj velikokrt poišče neodvisno množico, ki je iste velikosti, kot največja neodvisna množica dobljena s CLP. Neodvisna množica, dobljena z *lokalnim_iskanjem*, največkrat odstopa za eno vozlišče, od tiste dobljene s CLP, v veliko primerih se kot omenjeno ti popolnoma ujemata, najredkeje pa se razlikujeta za 3 vozlišča.



Slika 3: Odstopanja rešitev lokalnega iskanja do CLP.

Kot omenjeno, je maksimalne neodvisne množice z najvišjimi močmi vračal CLP, s povprečno močjo največje neodvisne množice 9. Druge največje množice je vračalo $lokalno\_iskanje$, katerega množice so imele v povprečju 8 vozlišč, za eno več od povprečja moči neodvisnih množic najdenih z algoritmom $nakljucni\_MIS$. Rezultati so vidni na spodnjem grafu.

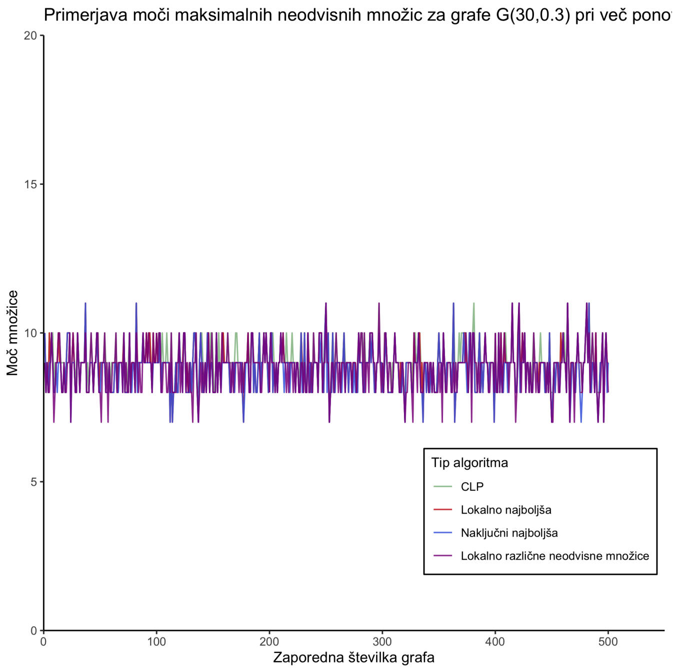


Slika 4: Povprečne vrednosti rezultatov posameznih algoritmov

3.3.2 Analiza izboljšanih algoritmov na grafih $G(30, 0.3)$

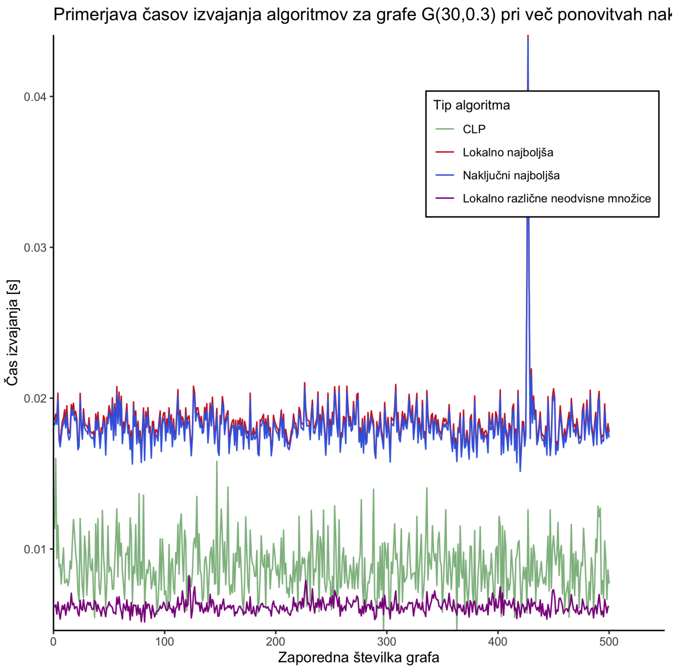
V nadaljevanju sva poskusila rezultat dobljen z $lokalnim\_iskanjem$ in algoritmom *nakljucni_MIS* nekoliko izboljšati. Najprej sva za rešitev dobljeno z $nakljucni\_MIS$ vzela najboljšo izmed dvajsetih simulacij za vsak graf, kar je na spodnjem grafu označeno z *Naključno najboljša*. Iz te dobljene neodvisne množice sva potem izhajala pri $lokalnem\_iskanju$ in rešitev označila kot *Lokalno najboljša*. Rešitev dobljena s CLP je še zmeraj enaka, sva pa dobila še maksimalne neodvisne množice na način, da sva algoritem $lokalno\_iskanje$

pognala na pet različnih začetnih neodvisnih množicah, dobljenih z $nakljucni\_MIS$. Na grafu je ta rešitev označena z *Lokalno različne neodvisne množice*. Na spodnjem grafu je razvidno, da sva na ta način dobila boljše rezultate kot z eno ponovitvijo $lokalnega\_iskanja$, so pa še zmeraj rešitve dobljene s CLP najboljše, saj z le-tem v kolikor najdemo kakšno rešitev, gre za največjo neodvisno množico.



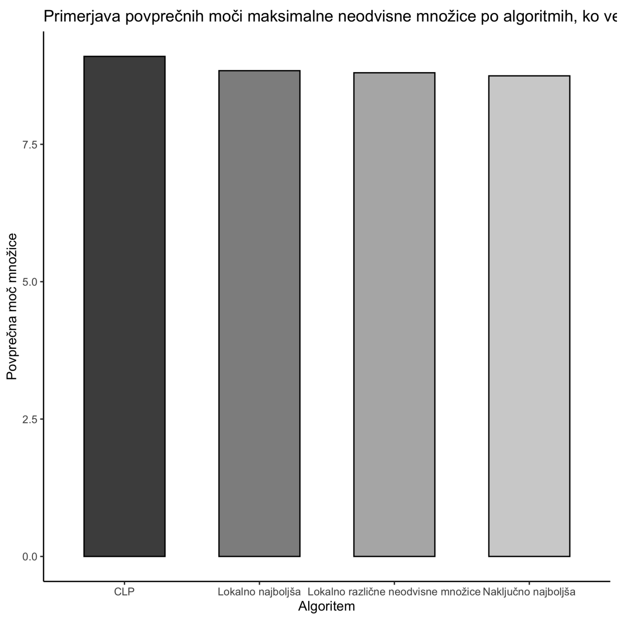
Slika 5: Vrednosti rezultatov izboljšanih posameznih algoritmov

Za zgoraj opisane postopke sva nato izmerila še čase izvajanja. Časovno najbolj učinkovit algoritem je ponovno variacija $lokalnega\_iskanja$, kjer le-tega poženeva na pet različnih začetnih neodvisnih množicah. Sedaj je čas izvajanja tega algoritma nekoliko večji, vendar z njim dobimo rešitve, ki se zelo približajo rešitvam CLP. Če bi število začetnih neodvisnih množic še povečala, bi bili rezultati še boljši, vendar bi s tem naraščala tudi časovna zahtevnost. Vmes se pojavijo tudi kakšni izraziti skoki, ki pa jih lahko pripišemo slučaju, saj je čas izvajanja zelo odvisen od vrste grafa, na katerem se algoritem izvede. Ker grafe generirava naključno je prav to vzrok za opažene osamelce. Na drugi strani imamo sedaj priredbi algoritmov $nakljucni\_MIS$, ko le-tega poženeva dvajsetkrat in vzamemo najboljšo rešitev ter algoritem še eno variacijo algoritma $lokalno\_iskanje$, kjer za začetno neodvisno množico vzameva prej omenjeno (najboljšo) rešitev iz $nakljucni\_MIS$. Izkaže se, da imata prav ta dva postopka iskanja neodvisnih množic največjo časovno zahtevnost. Zanimivo pa je dejstva, da kljub temu vrneta neodvisne množice najmanjših moči. Se pravi je ta pohlepen pristop, da bi od samega začetka $lokalno\_iskanje$ izvajala na čim večji množici neučinkovit. Zaključimo lahko, da z večanjem začetne neodvisne množice algoritem $lokalno\_iskanje$ vrača slabše rešitve, z večjim številom različnih začetnih neodvisnih množic pa $lokalno\_iskanje$ vrne zelo primerljive rezultate kot CLP.



Slika 6: Časovne zahtevnosti izboljšanih posameznih algoritmov

Da bi se res transparentno prepričali, da v tem primeru najboljše rešitve v povprečju poiščeva z *Lokalno različne neodvisne množice*, ko naredimo algoritem *lokalno_iskanje* na različnih začetnih neodvisnih množicah, sva izračunala povprečne moči, ki jih vračajo algoritmi. Na spodnjem grafu lahko to očitno opazimo.

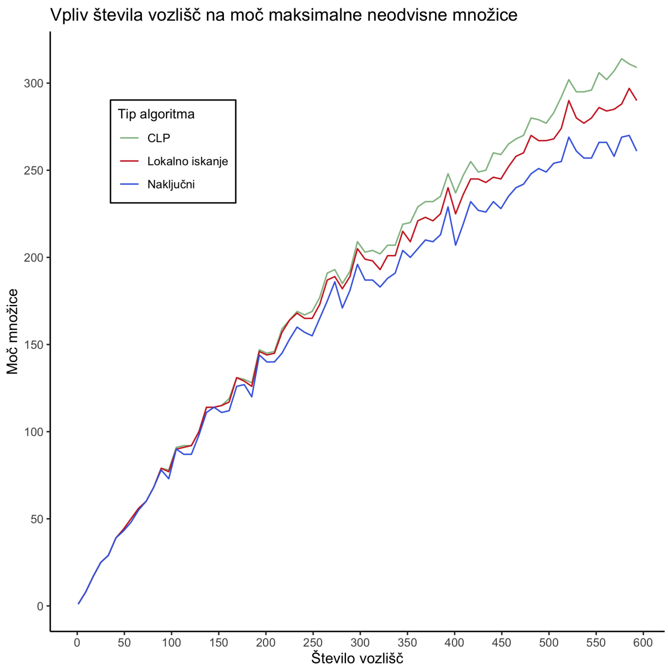


Slika 7: Povprečne vrednosti rezultatov izboljšanih posameznih algoritmov

3.3.3 Primerjava algoritmov na grafih s konstantno verjetnostjo p in spremenljivim številom vozlišč n

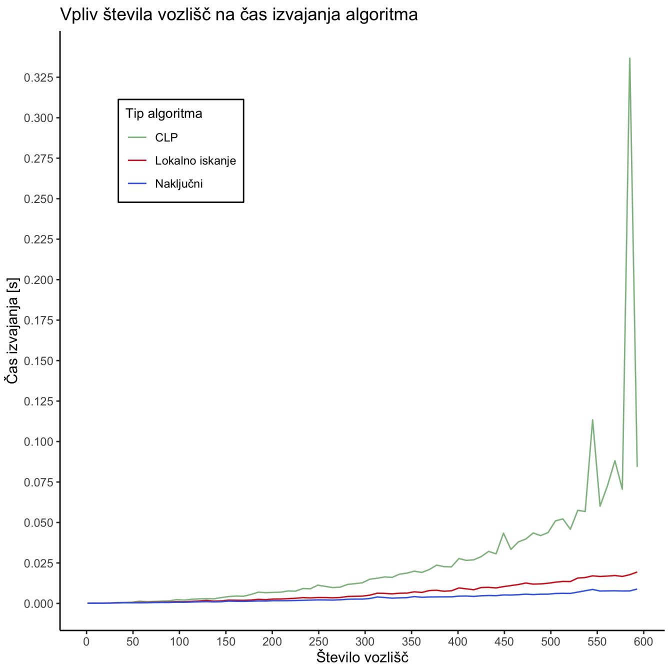
Drugo skupino grafov sva, tako kot prvo, generirala z Erdős-Rényijevim $G(n, p)$ modelom, vendar pri tem spreminjala število vozlišč v grafih, verjetnost pa ohranila konstantno $p = 0.005$. Algoritme sva izvajala na grafih velikosti od 1 do 600 vozlišč. Ker je verjetnost konstantna in na rezultate nima precejšnjega vpliva, sva si lahko izbrala grafe na večjem številih vozlišč ter pridobila več podatkov za bolj natančno analizo rezultatov.

Na spodnjem grafu si lahko ogledamo, kako se s povečevanjem števila vozlišč, spreminja moč maksimalnih neodvisnih množic. Pri konstantni verjetnosti p, se moč maksimalne neodvisne množice, v odvisnosti od moči množice vozlišč V, povečuje skoraj linearno, vendar moč maksimalnih neodvisnih množic dobljenih s CLP narašča hitreje od tistih pridobljenih z $nakljucni\_MIS$ in $lokalnim\ iskanjem$. Opazimo tudi, da so rešitve vseh treh algoritmov v začetku, ko jih izvajamo na manjših grafih zelo primerljive kar smo opazili tudi v prejšnjem razdelu. Z večanjem števila vozlišč v grafu G vidno narašča tudi razlika med močmi najdenih neodvisnih množic. Razkorak je prvič bolj izrazit na grafu s približno 150 vozlišči. Lahko bi rekli da "napaka"narašča proporcionalno z rastjo množice vozlišč V.



Slika 8: Velikosti neodvisnih množic v odvisnosti od števila vozlišč n

Na spodnjem grafu ponovno opazimo največjo slabost iskanja maksimalne neodvisne množice s CLP, tj. časovno zahtevnost. Čas, porabljen za izvajanje CLP, je v odvisnosti od števila vozlišč v množici V naraščal eksponentno. Časovna zahtevnost algoritmov $nakljucni\_MIS$ in $lokalnega\_iskanja$ je prav tako naraščala, vendar občutno počasneje, kot pri CLP.



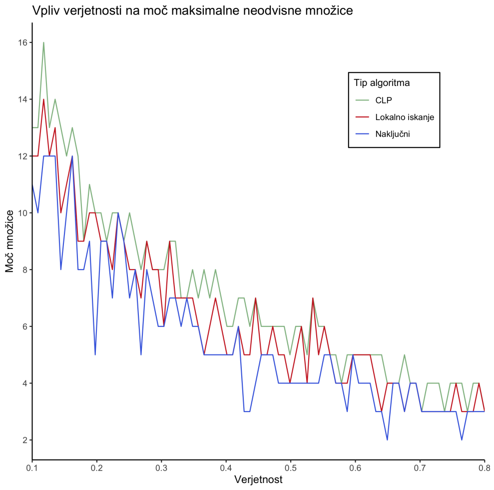
Slika 9: Spreminjanje časovne zahtevnosti algoritmov v odvisnosti od števila vozlišč n

3.3.4 Primerjava algoritmov na grafih s konstantim številom vozlišč n in s spremenljivo verjetnostjo p

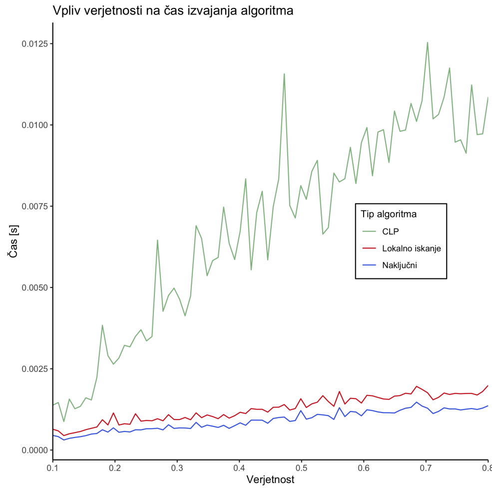
Tudi zadnjo skupino grafov sva generirala z Erdős-Rényijevim $G(n, p)$ modelom, pri tem pa ohranila konstantno velikost množice vozlišč V in spreminjala verjetnosti p. Verjetnosti p sva dobila s klicom funkcije $numpy.linspace(0.1, 0.8, 80)$.

Če je s povečevanjem velikosti množice V naraščala tudi velikost maksimalne neodvisne množice, opazimo, da je v primeru povečevanja verjetnosti p vzorec nasproten, saj velikost maksimalne neodvisne množice pada, ko povečujemo verjetnost p. Razlog za to se najverjetneje skriva v vplivu verjetnosti na povezave grafa G. Večja kot je verjetnost p, večje bo število povezav v grafu, kar je za velikost neodvisne množice slabo, saj so verjetnosti, da so vozlišča v grafu sosednja, večje. Torej velikost maksimalne neodvisne množice ni odvisna le od števila vozlišč v grafu, ampak tudi od števila povezav. Ugotovitve so opazne na spodnjem grafu.

Tako kot velikost maksimalne neodvisne množice, je od števila povezav v grafu (oz. verjetnosti p), odvisna tudi časovna zahtevnost algoritmov. Časovna zahtevnost algoritmov ob večanju verjetnosti p (in posledično števila povezav v grafu) se obnaša podobno, kot se je obnašala pri večanju števila vozlišč v grafu G. Časovna zahtevnost vseh treh algoritmov narašča z verjetnostjo p, vendar pri CLP hitreje kot pri algoritmih $nakljucni\_MIS$ in $lokalno\_iskanje$. Graf (b) to nazorno prikazuje in potrjuje.



(a) Vpliv verjetnosti na velikost neodvisnih množic



(b) Vpliv verjetnosti na časovno zahtevnost algoritmov

4 Zaključek

Iz analize rezultatov lahko zaključimo, da v primeru generiranja grafov z *Erdős-Rényijevim* $G(n, p)$ modelom, na velikost maksimalne neodvisne množice v danem grafu vplivata tako število vozlišč v grafu, kot tudi število povezav. Kot alternativo iskanju maksimalne neodvisne množice s CLP-jem, sva uspela implementirati algoritem $nakljucni\_MIS$ in ga kasneje izboljšati z *lokalnim_iskanjem*. Čeprav so rešitve CLP malenkost boljše v večini primerov, sta $nakljucni\_MIS$ in *lokalno iskanje* boljša za grafe z večjim številom vozlišč, ali pa z večjim številom povezav, saj sta bolj časovno učinkovita. Morda se nam za zelo veliko vozlišč vseeno splača rešiti problem s CLP, saj kot prej omenjeno "napaka" proporcionalno narašča z večanjem števila vozlišč v grafu G. Zelo pa sva bila navdušena nad dejstvom, da sva z izboljšanim *lokalnim_iskanjem* v primeru grafa $G(30, 0.3)$ uspela poiskati maksimalne neodvisne množice večje moči kot s CLP.

Literatura

[1] Gary Miller. *Lecture 32: Luby's Algorithm for Maximal Independent Set*, dostopno na http://www.cs.cmu.edu/afs/cs/academic/class/15750-s18/ScribeNotes/lecture32.pdf.

[2] Diogo Andrade, Mauricio G. C. Resende. *Fast Local Search for the Maximum Independent Set Problem.* Conference Paper in Journal of Heuristics, May 2008, dostopno na https://www.researchgate.net/publication/221131653_Fast_Local_Search_for_the_Maximum_Independent_Set_Problem.

[3] *Maximal independent set*, v: Wikipedia: The Free Encyclopedia,[ogled 6. 1. 2022], dostopno na https://en.wikipedia.org/wiki/Maximal_independent_set.

[4] *Independent set (graph theory)*, v: Wikipedia: The Free Encyclopedia,[ogled 6. 1. 2022], dostopno na https://en.wikipedia.org/wiki/Independent_set_(graph_theory).