# UČINKOVITOST ALGORITMOV ZA ISKANJE NAJVEČJE NEODVISNE MNOŽICE VOZLIŠČ V GRAFIH

MATEJ ŠKERLEP

Fakulteta za matematiko in fiziko
Univerza v Ljubljani

Problem največje neodvisne množice vozlišč je zanimiv problem iz teorije grafov, z izjemno uporabno vrednostjo pri določanju zaporedja v molekulah deoksiribonukleinskih kislin. V članku je za izhodišče vzeta formulacija problema v obliki celoštevilskega linearnega programa, ki je nato primerjan s svojo relaksacijo in algoritmom za lokalno iskanje na grafu. Primerjava je narejena na podlagi dobljenih množic vozlišč, ki jih algoritmi vračajo, in na podlagi časovne zahtevnosti posameznega algoritma. Analize so izvedene na naključnih grafih z do 100 vozlišči pri različnih verjetnostih povezave med dvema vozliščema. Za analizo časovne zahtevnosti je število vozlišč grafov povečevano do 2401. Na koncu je narejena še analiza na Petersenovem grafu in $n$-dimenzionalnih hiperkockah za $1 \leq n \leq 12$. Na slednjih se za učinkovito izkaže tudi relaksacija celoštevilskega linearnega programa, kar omogoča ustrezno primerjavo časovne zahtevnosti vseh treh algoritmov.

**THE EFFICIENCY OF ALGORITHMS FOR FINDING THE MAXIMUM INDEPENDENT SET OF VERTICES IN GRAPHS**

Finding the maximum independent vertex set is an intriguing graph theoretical problem that is extremely useful in determining sequences in deoxyribonucleic acids. In this paper, the problem is defined as an integer linear program and contrasted to the relaxation of it. The assessments of the integer linear program, its relaxation, and the algorithm for local search in a graph are based on the output set's cardinality and the run-time analyses. Experiments are performed on random graphs with up to 100 vertices, which have a different probability of an edge connecting two arbitrary vertices. For analysis of the algorithm's run-time, the number of vertices is increased to 2401. Finally, tests are carried out on the Petersen graph and $n$-dimensional hypercubes for $1 \leq n \leq 12$. On the latter, even the relaxation of the integer linear program successfully finds the maximum independent set, which enabled making insightful comparisons of run-time.

## 1. Uvod

V teoriji grafov največja neodvisna množica vozlišč predstavlja enega izmed klasičnih problemov, ki ga je Baker [1] dokazal za NP-težkega [2]. Cilj problema je v poljubnem grafu najti podmnožico vozlišč, ki ne vsebuje sosednjih vozlišč in ima največjo moč [3]. Problem največje množice vozlišč ima uporabno vrednost pri dokazovanju časovne zahtevnosti teoretičnih matematičnih modelov [4], kot tudi pri reševanju mnogih realnih optimizacijskih problemov v genetiki [5, 6] (cit. v [2]).

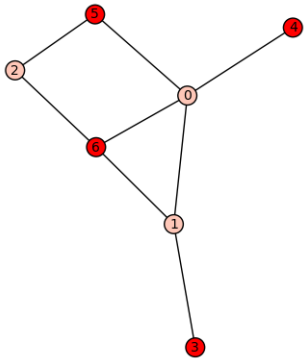
V članku bomo zgoraj omenjeni problem definirali kot celoštevilski linearni program in naredili primerjavo z njegovo relaksacijo. Ugotovili bomo razlog, zakaj se relaksacija v mnogih primerih izkaže za neučinkovito. Z uporabo programskega jezika *Sage* bomo implementirali oba linearna programa in naredili analize tako glede uspešnosti določanja iskane množice, kot tudi glede časovne zahtevnosti posameznega algoritma. Rezultate bomo primerjali z algoritmom za lokalno iskanje v neusmerjenem grafu. Poleg testov na naključnih grafih bomo analize izvedli tudi na Petersenovem grafu in hiperkockah.

## 2. Definicije pojmov

Za začetek si oglejmo formalno definicijo problema in kako ga lahko predstavimo kot linearni program.

**Definicija 1 ([7]).** Naj bo $G = (V, E)$ neusmerjen graf in $J \subseteq V(G)$. Množica $J$ je *neodvisna*, če ne vsebuje sosednjih vozlišč. Formalno, če za $\forall v, u \in V$, $uv \in E$ velja $v \in J \implies u \notin J$. Neodvisni množici na grafu $G$ z največjo močjo pravimo *največja neodvisna množica*. V nadaljevanju jo bomo označevali z $I$.

**Primer 1.** Za nazornejšo predstavo zgornje definicije si oglejmo sliko 1, ki prikazuje preprost primer največje neodvisne množice na grafu s 7 vozlišči. Zlahka se lahko prepričamo, da množica vozlišč $\{3, 4, 5, 6\}$ predstavlja največjo množico neodvisnih vozlišč.




**Slika 1.** Primer največje neodvisne množice na grafu s sedmimi vozlišči (označena rdeče).

### 2.1 Celoštevilski linearen program (CLP)

Tako kot v definiciji 1 bo v celotnem članku največja neodvisna množica označevana z $I$. Oglejmo si, kako lahko problem največje neodvisne množice definiramo kot celoštevilski linearni program. V nadaljevanju razdelka 2.1 in v razdelku 2.2 se bomo oprli na formulacijo Matouseka in Gärtnerja [8, str. 39–40].

Vsakemu vozlišču $v \in V$ priredimo spremenljivko $x_v$ z vrednostjo 1 ali 0, glede na to ali je vsebovana v največji neodvisni množici ali ne;

$$x_v = \begin{cases} 1; & v \in I \\ 0; & v \notin I \end{cases}.$$

Pogoj $\forall uv \in E$: $x_u + x_v \leq 1$ nam tako zagotovalja, da ima vsaka povezava $uv \in E$ največ eno vozlišče v množici $I$ [8]. Vrednost kriterijske funkcije spodnjega CLP je enaka moči množice $I$, v njej pa so vsa vozlišča $v$, za katera je $x_v = 1$ [8]. Tako dobimo, enako kot Matousek in Gärtner [8], naslednji linearni program.

$$\begin{aligned} \max \quad & \sum_{v \in V} x_v \\ \text{p.p.} \quad & x_u + x_v \leq 1, \quad \forall uv \in E \\ & x_v \in \{0, 1\}, \quad \forall v \in V \end{aligned}$$

### 2.2 Relaksacija celoštevilskega linearnega programa

V primeru relaksacije zgornjega celoštevilskega linearnega programa bomo omejitev vrednosti $x_v$ na 0 ali 1 sprostili na vse vrednosti znotraj intervala teh dveh števil. Pogoj $x_v \in \{0, 1\}$ zamenjamo s pogojem $0 \leq x_v \leq 1$ in tako lahko zapišemo spodnji linearni program [8].

$$\begin{aligned} \max \quad & \sum_{v \in V} x_v \\ \text{p.p.} \quad & x_u + x_v \leq 1, \quad \forall uv \in E \\ & 0 \leq x_v \leq 1, \quad \forall v \in V \end{aligned}$$

V tem primeru dobimo trivialno rešitev $x_v = \frac{1}{2}$ za $\forall v \in V$ in pa vrednost kriterijske funkcije $\frac{|V|}{2}$ [8], kar pa nam o tem, katera vozlišča so v množici $I$, ne pove ničesar. Iz minimalne vrednosti kriterijske funckije lahko torej sklepamo, da mora vedno veljati $\sum_{v \in V} x_v \geq \frac{|V|}{2}$ [8].

**Primer 2 ([8]).** Oglejmo si primer na polnem povezanem grafu z $n$ vozlišči. Jasno je, da množica $I$ vsebuje zgolj eno vozlišče. Po drugi strani pa nam bo relaksacija CLP vedno vračala moč vsaj $\frac{n}{2}$, kar bo enako tudi vrednosti njene kriterijske funkcije. Tako lahko opazimo, da se relaksacija CLP precej razlikuje od klasičnega CLP in, kot bomo videli v nadaljevanju, je na gosto povezanih grafih pogosto neuporabna.

V izogib trivialni rešitvi bodo v množico $I$ dodana zgolj tista vozlišča, ki bodo imela vrednost $x_v > \frac{1}{2}$. S tem zagotovimo, da bo v $I$ vsebovano največ eno vozlišče vsake povezave, s čimer bomo dobili neodvisno množico in torej dopustno rešitev. V nadaljevanju bomo videli, da se pogosto zgodi, da je prav rešitev $x_v = \frac{1}{2}$ za $\forall v \in V$ optimalna in posledično (glede na zgoraj opisani pogoj $x_v \in I \iff x_v > \frac{1}{2}$) bo množica $I$ prazna.

## 3. Algoritmi

### 3.1 Izbira programskega jezika in knjižnic ter splošen opis kode

Algoritmi so napisani v programskem jeziku *Sage* [9], ki temelji na programskem jeziku *Python*. Uporabili smo module *random* za generiranje naključnega nabora vozlišč v algoritmu lokalnega iskanja, *time* za merjenje časa za izvedbo posameznega algoritma in *csv* za shranjevanje rezulatov analiz v csv obliki. Analize so bile izvedene na spletni platformi *CoCalc* [10].

V kodi so sprva napisane vse tri funkcije `clp` za celoštevilski linearni program, `rclp` za relaksacijo celoštevilskega linearnega programa in `lokalno_iskanje` za lokalno iskanje. Nato je bila za namene testiranja definirana funkcija `testiraj`, ki na naključnem grafu (kot argumente grafa določimo število vozlišč in verjetnost povezave med dvema vozliščema) izvede vse tri zgoraj omenjene funkcije in kot rezultat vrne seznam izhodnih podatkov vseh treh funkcij. Na koncu je dodana še funkcija `izpis_csv`, ki rezultate testiranj zapiše v csv datoteteko. Posamezna vrstica v csv datoteki predstavlja izhodne podatke vseh treh algoritmov za graf z določenim številom vozlišč. Poleg testov na naključnih grafih so bili le-ti nato izvedeni še na Petersenovem grafu in nekaterih hiperkockah.

Za namene analize časovne zahtevnosti so v vseh treh funkcijah uporabljeni klici `time.time()`, čas celotne izvedbe posameznega algoritma pa lahko torej dobimo kot razliko med končnim in začetnim časom (t.j. `cas = kon - zac`).

### 3.2 Celoštevilski linearni program

Sprva si oglejmo algoritem celoštevilskega linearnega programa. Funkcija `clp` kot vhodni argument vzame graf. Celoštevilski linearni program najprej definiramo z `MixedIntegerLinearProgram`, nato vsakemu vozlišču $v$ priredimo binarno spremenljivko `vozlisca(v)` z vrednostmi 0 ali 1. Kriterijsko funkcijo definiramo s klicem funkcije `set_objective`, omejitev za vsaki dve povezani vozlišči pa z `add_constraint`. Optimalno vrednost kriterijske funkcije dobimo s `CLP.solve()`, kar je hkrati tudi moč največje neodvisne množice. Vrednost $x_v$ posameznega vozlišča dobimo s `CLP.get_values(vozlisca)`. V iskano največjo množico neodvisnih vozlišč `sez` torej dodamo vozlišča z vrednostjo $x_v$ enako 1. Kot izhodni podatek na koncu algoritem vrne seznam oblike: moč največje neodvisne množice, seznam vozlišč v tej množici in čas izvedbe algoritma. Funkcija `int()` nam število z decimalno vejico zapiše v celoštevilski obliki, kar je uporabljeno zaradi težav Excela pri branju podatkov. Analize izhodnjih rezultatov funkcij in vizualizacije so narejene v Excelu.

```
def clp(graf):
    zac = time.time()

    CLP = MixedIntegerLinearProgram(maximization = True)
    vozlisca = CLP.new_variable(binary = True)
    CLP.set_objective(sum([vozlisca[v] for v in graf.vertices()]))
    for u,v in graf.edges(labels = False):
        CLP.add_constraint(vozlisca[u] + vozlisca[v] <= 1)

    moc_max = CLP.solve()
    vozlisca = CLP.get_values(vozlisca)

    sez = [k for k, v in vozlisca.items() if v == 1.0]

    kon = time.time()
    cas = kon - zac
    return [int(moc_max), sez, cas]
```

### 3.3 Relaksacija celoštevilskega linearnega programa

Implementacija relaksiranega CLP je podobna kot implementacija CLP, zato v nadaljevanju obrazložimo zgolj glavne razlike. Pri relaksaciji CLP vrednosti $x_v$ omejimo s `set_max` oz. `set_min`. Vrednost kriterijske funkcije nam o moči množice ne pove veliko, zato z argumenti iz razdelka 2.2 dodajamo v največjo množico neodvisnih vozlišč (v algoritmu označena s `sez`) zgolj vozlišča, za katera je $x_v > \frac{1}{2}$. Funkcija sprejme in vrne enake argumente kot `clp`.

```
def rclp(graf):
    zac = time.time()

    RCLP = MixedIntegerLinearProgram(maximization = True)
    vozlisca = RCLP.new_variable(real = True)
    RCLP.set_max(vozlisca,1)
    RCLP.set_min(vozlisca,0)
    RCLP.set_objective(sum([vozlisca[v] for v in graf.vertices()]))
    for u,v in graf.edges(labels = False):
        RCLP.add_constraint(vozlisca[u] + vozlisca[v] <= 1)

    RCLP.solve()
    vozlisca = RCLP.get_values(vozlisca)

    sez = [k for k, v in vozlisca.items() if v > 0.5]
    moc_max = len(sez)

    kon = time.time()
    cas = kon - zac
    return [int(moc_max), sez, cas]
```

### 3.4 Lokalno iskanje

Alternativna možnost zgornjima dvema algoritmoma je lokalno iskanje, katerega ideja je sledeča. Začnemo s prazno množico $I$ v katero dodamo naključno izbrano vozlišče. Nato po naključnem izboru vozlišč v $I$ dodamo tista, katerih sosedje niso v množici $I$. To zanko ponovimo poljubno mnogokrat in nato vzamemo tisto množico $I$, ki ima največjo moč.

```
def lokalno_iskanje(graf, st_ponovitev):
    moc_max = 0
    max_mnozica_neod = set()
    zac = time.time()
    for i in range(st_ponovitev):
        mnozica_neod = set()
        mnozica_neod.add(graf.random_vertex())
        for v in random.sample(graf.vertices(), len(graf.vertices())):
            if not any(u in mnozica_neod for u in graf[v]):
                    mnozica_neod.add(v)
        if len(mnozica_neod) > moc_max:
            max_mnozica_neod = mnozica_neod
            moc_max = len(mnozica_neod)
    kon = time.time()
    cas = kon - zac
    return[moc_max, max_mnozica_neod, cas]
```

## 4. Rezultati

V namen analize učinkovitosti algoritmov so bile vse tri zgornje funkcije izvedene na istih naključnih grafih. Kot argumente generiranja grafov so bile vzete vse možne kombinacije števila vozlišč (1 do 100) in verjetnosti povezave dveh vozlišč $\left(\frac{1}{10}, \frac{3}{10}, \frac{5}{10}, \frac{7}{10}\right.$ in $\left.\frac{9}{10}\right)$. Pri tem je bilo število ponovitev lokalnega iskanja nastavljeno na 50.

Primerjane so moči največjih neodvisnih množic vseh treh algoritmov. Generalno gledano za vse rezultate testiranj velja, da je moč množice dobljene s CLP vedno večja ali enaka moči množice dobljene z lokalnim iskanjem. Algoritem relaksacije CLP od določenega števila vozlišč dalje (pri velikih verjetnostih povezave med vozliščema že od 3 vozlišč naprej), skoraj vedno vrača prazno množico, kar pomeni, da je očitno optimalna vrednost kriterijske funkcije ravno, ko imajo vsa vozlišča vrednost $x_v$ enako $\frac{1}{2}$.

### 4.1 Analiza algoritmov na naključnih grafih

#### 4.1.1 Primerjava moči množic vseh treh algoritmov

Za začetek si v tabeli 1 oglejmo, do katerega števila vozlišč grafa dobimo enake moči dobljenih množic vseh treh algoritmov.

| Verjetnost povezave | Enakost vseh treh algoritmov |
|---|---|
| 0,1 | do 18 vozlišč |
| 0,3 | do 7 vozlišč |
| 0,5 | do 4 vozlišč |
| 0,7 | do 4 vozlišč |
| 0,9 | do 2 vozlišč |

**Tabela 1.** Največje število vozlišč do katerega se moči največje neodvisne množice posameznega algoritma ujemajo.

Opazno je precejšnje zmanjševanje. Vidimo, da v primeru gosto povezanega grafa, dobimo enakost zgolj še na 2 vozliščih. Glavni razlog za neenakost je relaksacija CLP, za katero je za grafe z večjim številom vozlišč ponavadi optimalna rešitev, ko imajo vsa vozlišča vrednost $\frac{1}{2}$ in posledično je moč množice $I$ enaka 0. Slednji rezultat je povsem pričakovan, saj se algoritem relaksacije CLP na grafih z velikimi verjetnostmi povezave obnaša zelo podobno kot na polnem grafu, kar je bilo opisano v primeru 2.

#### 4.1.2 Primerjava moči množic CLP in lokalnega iskanja

Bolj zanimivo si je ogledati enakost med močmi množic, dobljenih s CLP in lokalnim iskanjem. Tabela 2 prikazuje, do katerega števila vozlišč dobimo enakosti moči in največjo razliko med močema množice $I$, dobljene s CLP in lokalnim iskanjem na celotnem vzorcu grafov od 1 do 100 vozlišč pri posameznih verjetnostih.

| **Verjetnost povezave** | **Enakost CLP in lokalnega iskanja** | **Največja razlika v močeh** |
|---|---|---|
| 0,1 | do 33 vozlišč | 4 |
| 0,3 | do 26 vozlišč | 3 |
| 0,5 | do 35 vozlišč | 2 |
| 0,7 | do 38 vozlišč | 1 |
| 0,9 | do 58 vozlišč | 1 |

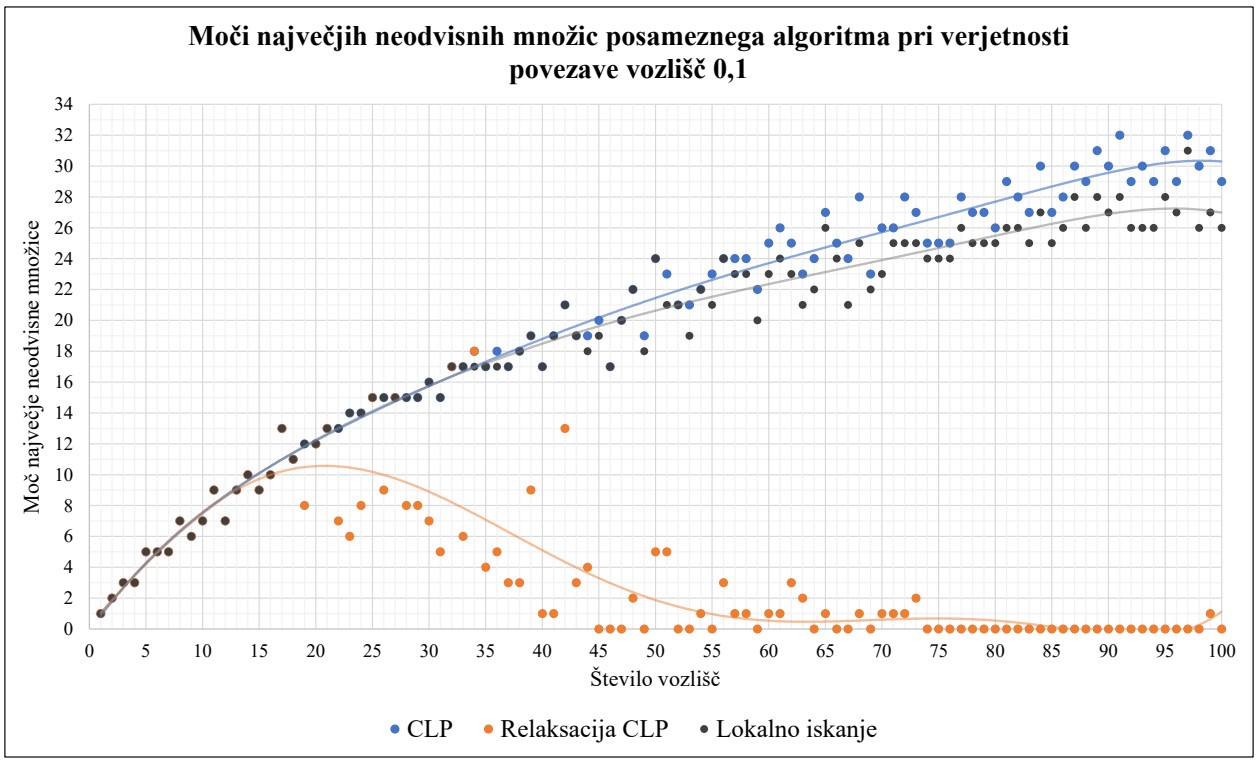
**Tabela 2.** Največje število vozlišč, do katerega se moči največje neodvisne množice CLP in lokalnega iskanja ujemata, ter največja razlika v močeh obeh množic na grafih z do 100 vozlišči.

Opazimo lahko, da je najmanjše ujemanje pri nizkih verjetnostih povezave. Pri 10 % verjetnosti dobivamo enakost do 33 vozlišč, sicer pa je nato do 66 vozlišč razlika v močeh največ 2. Vrednost razlike se pri večjih grafih (od vključno 91 vozlišč naprej) poveča na 4, a slednje vrednosti nikoli ne preseže. Pri grafih z verjetnostjo 30 % se meja enakosti pomakne nekoliko navzdol, a se hkrati meja, ko je razlike v močeh največ 2, pomakne na 76 vozlišč. Pri višjih verjetnostih se začne meja enakosti precej povečevati, poleg tega pa tudi manjšati maksimalna razlika med močmi dobljenih množic. Pri 50 % verjetnosti povezave je razlika v močeh največ dva, pa še to zgolj v treh primerih. Pri verjetnosti povezave 70 % se moči množic razlikujeta za 1 le v 15 primerih, pa še to z izjemo dveh primerov od grafa s 65 vozlišči naprej. Zgolj pet primerov, ko se vrednosti moči razlikujeta za 1, se pojavi pri verjetnosti povezave 90 %, na vseh ostalih grafih pa dobimo enakosti.

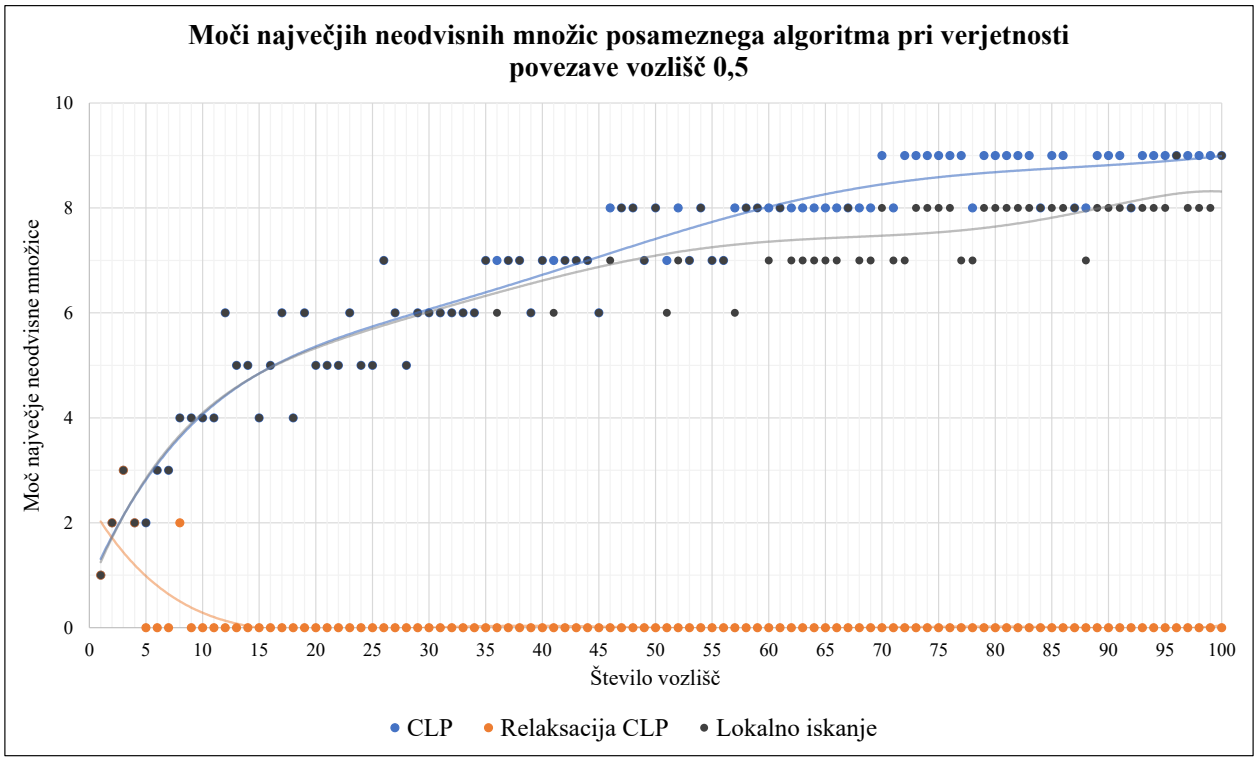
#### 4.1.3 Grafična predstavitev rezultatov

Za lažjo predstavo zgoraj opisanih rezultatov analize smo le-te prikazali tudi na spodnjih dveh grafih (na slikah 2 in 3), ki prikazujeta moči dobljenih množic v odvisnosti od števila vozlišč. Na grafih je prikazana tudi aproksimacija gibanja vrednostni s polinomom. Kot smo omenili že zgoraj, je za večje grafe opazen padec moči množice, dobljene z relaksacijo CLP, sploh v primerih z veliko verjetnostjo povezave dveh vozlišč grafa.

Zanimivo je, da z večanjem verjetnosti moč največje neodvisne množice CLP in lokalnega iskanja postaja konstantna na določenih intervalih števila vozlišč. Na sliki 3 je opazno, da CLP pri 60 vozliščih grafa ali več zelo pogosto vrača množice z enim vozliščem več kot lokalno iskanje. Z večanjem števila vozlišč, se moč množice tudi precej počasneje povečuje in pri bolj gosto povezanih grafih celo nikoli ne preseže vrednosti 5.




**Slika 2.** Moči največjih neodvisnih množic dobljenih s posameznim algoritmom na naključnih grafih pri verjetnosti povezave 0,1.



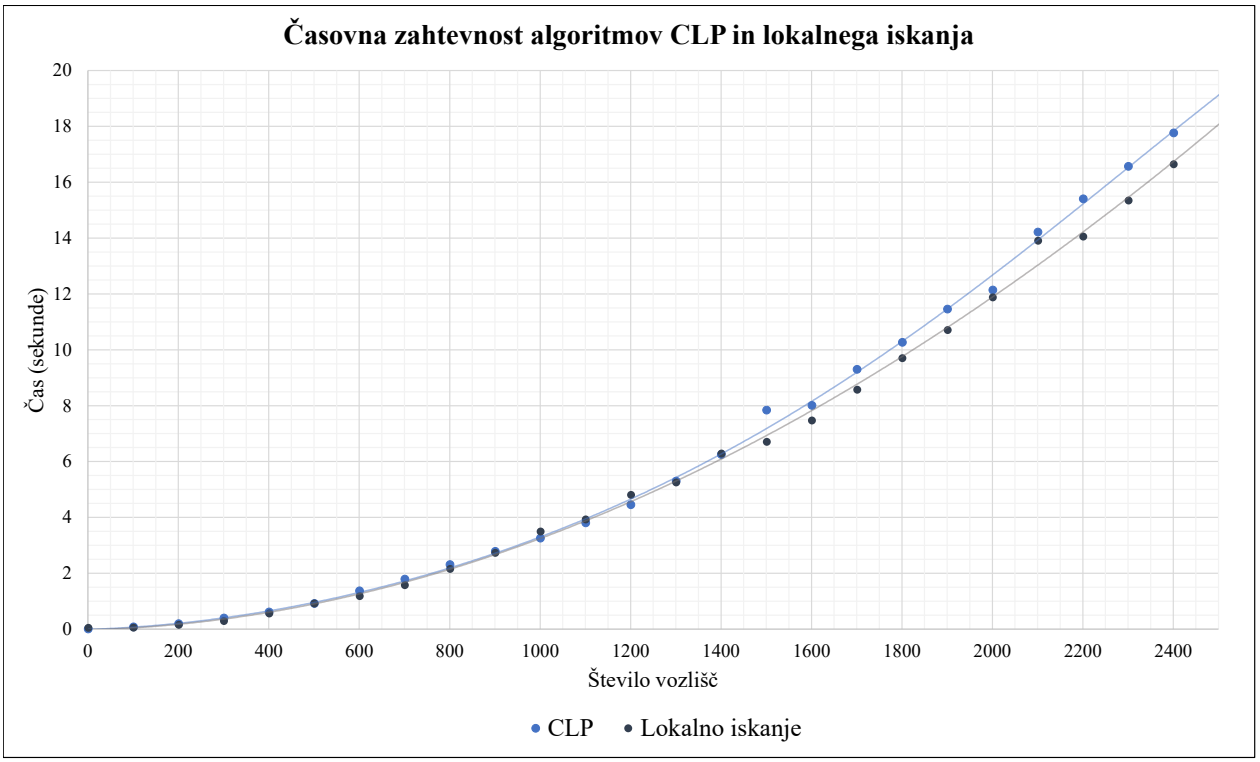

**Slika 3.** Moči največjih neodvisnih množic dobljenih s posameznim algoritmom na naključnih grafih pri verjetnosti povezave 0,5.

#### 4.1.4 Časovna zahtevnost

Pri analizi časovne zahtevnosti se bomo omejili zgolj na CLP in algoritem za lokalno iskanje. Časovne zahtevnosti relaksacije CLP ne bomo analizirali, saj algoritem za velike grafe skoraj vedno vrača prazno množico in nam čas, ki je potreben za to "neustrezno" rešitev, ne da dobre primerjave.

V namen analize časovne zahtevnosti algoritma lokalnega iskanja bomo slednjega izvedli na grafu s 30 % verjetnostjo povezave dveh vozlišč in 50 ponovitvami zunanje zanke. Neodvisna spremenljivka je torej število vozlišč, ki jo bomo povečevali s korakom 100 do končne velikosti grafa z 2401 vozlišči. Meritve časa so izvedene trikrat, nato pa je vzeto povprečje vseh treh časov pri posamezni velikosti grafa.

V omenjeni analizi je algoritem lokalnega iskanja v povprečju hitrejši od CLP za 4,68 %. Razlike v hitrosti se z večanjem števila vozlišč seveda povečujejo – pri 2201 vozlišču je lokalno iskanje hitrejše za 9,55 %. Na spodnjem grafu (prikazanem na sliki 4) so vidni tudi rezultati za oba algoritma in aproksimacijska polinoma.



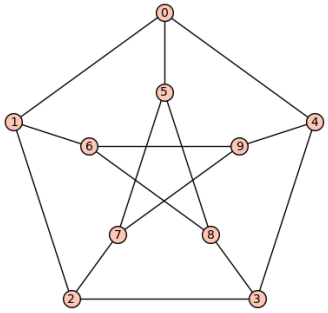

**Slika 4.** Časovna zahtevnost algoritmov CLP in lokalnega iskanja na naključnih grafih z 1 do 2401 vozlišči in verjetnostjo povezave 30 %.

### 4.2 Analiza algoritmov na Petersenovem grafu

V nadaljevanju si bomo ogledali učinkovitosti algoritmov na Petersenovem grafu.

**Definicija 2 ([11, 12]).** *Petersenov graf* $G$ je definiran kot graf vseh podmnožic moči 2 množice s 5 elementi, pri čemer sta dve množici sosednji natanko tedaj, ko sta disjunktni.

Anternativno ga lahko enolično karakteriziramo tudi kot 3-regularen graf z desetimi vozlišči, ki ne vsebuje trikotnikov in štirikotnikov (oziroma ima ožino 5) [11]. Petersenov graf je prikazan na sliki 5.




**Slika 5.** Petersenov graf.

CLP in lokalno iskanje vedno najdeta enako moč največje neodvisne množice, medtem ko so vozlišča v njej enaka le v 10 % poskusov. Relaksacija CLP konstantno vrača prazno množico, kar je pričakovano na podlagi zgornjih analiz na naključnih grafih. Ponovno je opazna tudi večja učinkovitost algoritma lokalnega iskanja, ki je bil v primerjavi s CLP hitrejši za 34,77 %. Časovne zahtevnosti in moči množice $I$ so podane v spodnji tabeli 3.

| **Algoritem** | **Moč množice $I$** | **Čas izvedbe algoritma** |
|---|---|---|
| CLP | 4 | 0.0106859 $s$ |
| Relaks. CLP | 0 | 0.0023558 $s$ |
| Lokalno iskanje | 4 | 0.0069702 $s$ |

**Tabela 3.** Moč največje neodvisne množice in časovna zahtevnost na Petersenovem grafu.

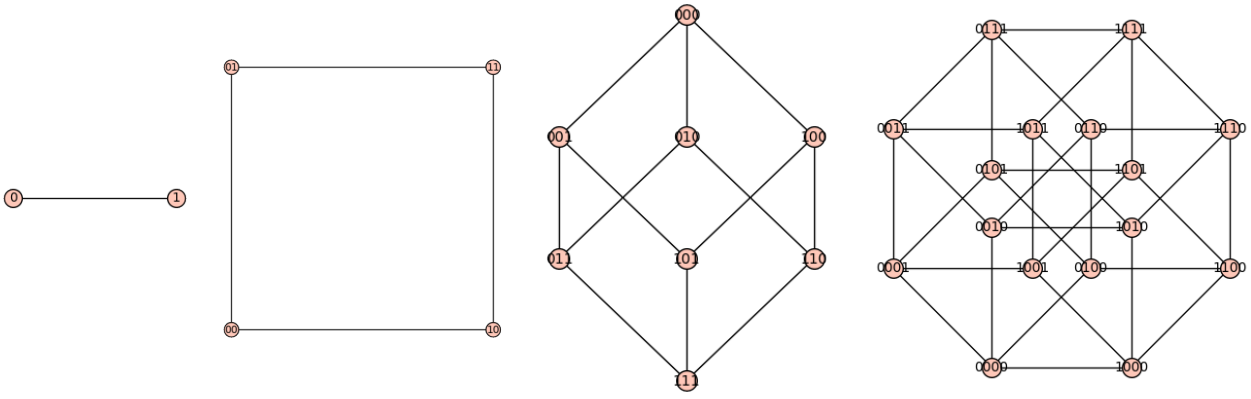
### 4.3 Analiza algoritmov na hiperkockah

**Definicija 3 ([13]).** Naj bo $K_2$ polni graf z dvema vozliščema. Potem lahko *n-dimenzionalno hiperkocko* $Q_n$ definiramo rekurzivno s pomočjo kartezičnega produkta grafov kot

$$Q_1 = K_2$$
$$Q_n = K_2 \,\square\, Q_{n-1},\ n \geq 2.$$

Alternativno lahko *n-dimenzionalno hiperkocko* $Q_n$ definiramo nerekurzivno kot graf, čigar vozlišča so označena z $n$-dimenzionalnimi dvojiškimi vektorji (tj. vektorji s koordinatami 0 ali 1) in sta v njem dve vozlišči povezani natanko tedaj, ko se njuna dvojiška vektorja razlikujeta v natanko eni koordinati.

Za lažjo predstavo definicije si oglejmo sliko 6, ki prikazuje *n-dimenzionalne hiperkocke* za $1 \leq n \leq 4$.




**Slika 6.** Hiperkocke $Q_n$ za $1 \leq n \leq 4$, ter pripradajoče binarne oznake vozlišč.

Pri analizi vseh treh algoritmov na hiperkockah smo pogledali največje množice neodvisnih vozlišč grafa in časovne zahtevnosti algoritmov na hiperkockah $Q_n$ za $2 \leq n \leq 7$. V tabeli 4 lahko opazimo, da so vse tri množice $I$ ves čas enake in da je njihova moč enaka ravno polovici števila vseh vozlišč grafa. Slednji rezultat je seveda pričakovan, saj so hiperkocke dvodelni grafi. Zanimivo je učinkovito delovanje relaksacije celoštevilskega linearnega programa za razliko od zgornjih analiz na naključnih grafih in Petersenovem grafu.

| $n$ | **Število vozlišč grafa $Q_n$** | **CLP** | **Relaks. CLP** | **Lokalno iskanje** |
|---|---|---|---|---|
| **2** | 4 | 2 | 2 | 2 |
| **3** | 8 | 4 | 4 | 4 |
| **4** | 16 | 8 | 8 | 8 |
| **5** | 32 | 16 | 16 | 16 |
| **6** | 64 | 32 | 32 | 32 |
| **7** | 128 | 64 | 64 | 64 |

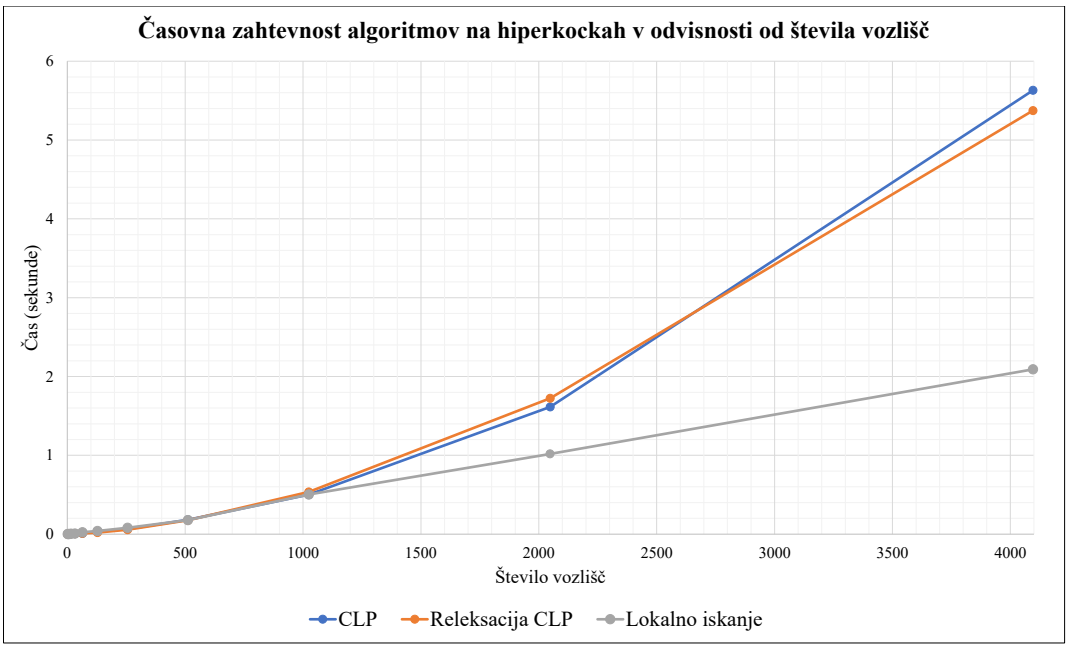
**Tabela 4.** Moč največje neodvisne množice vozlišč dobljene s posameznim algoritmom na hiperkockah.

### 4.3.1 Časovna zahtevnost algoritmov na hiperkockah

Ker so hiperkocke primer dvodelnih grafov, na katerih relaksacija CLP očitno daje pravilne rezultate, je smotrno na tem mestu narediti primerjavo časovne zahtevnosti vseh treh algoritmov. Analize so bile izvedene na hiperkockah $Q_n$ za $2 \leq n \leq 12$. V tabeli 5 so prikazane časovne zahtevnosti do $n = 5$ vozlišč, na grafu (prikazanem na sliki 7) pa so grafično predstavljeni še vsi rezultati.

| $n$ | **Število vozlišč grafa $Q_n$** | **CLP** | **Relaks. CLP** | **Lokalno iskanje** |
|---|---|---|---|---|
| **2** | 4 | 0.0027 | 0.0011 | 0.0033 |
| **3** | 8 | 0.0027 | 0.0036 | 0.0053 |
| **4** | 16 | 0.0056 | 0.0034 | 0.0386 |
| **5** | 32 | 0.0100 | 0.0280 | 0.0239 |

**Tabela 5.** Časovna zahtevnost algoritmov na hiperkockah (v sekundah).




**Slika 7.** Časovna zahtevnost algoritmov na hiperkockah $Q_n$ za $2 \leq n \leq 12$.

Na zgornjem grafu lahko opazimo, da se algoritem lokalnega iskanja konstantno spreminja v odvisnosti od števila vozlišč in da se do 1000 vozlišč algoritmi po časovni zahtevnosti ne razlikujejo prav dosti, nato pa začne prihajati do večjih odstopanj.

## 5. Zaključki

Na osnovi analiz lahko zaključimo, da je na naključnih grafih in pri Petersenovem grafu relaksacija CLP praktično neuporabna, saj prepogosto vrača prazne množice, kar pomeni, da je očitno optimalna vrednost kriterijske funkcije, ko imajo vsa vozlišča vrednost $x_v$, enaka $\frac{|V|}{2}$. Zanimivo je, da z večanjem verjetnosti moč največje neodvisne množice CLP in lokalnega iskanja postaja konstantna na določenih intervalih števila vozlišč.

Na Petersonovem grafu pričakovano CLP in lokalno iskanje vračata optimalen rezultat, relaksacija CLP pa ponovno odpove in vrača prazno množico. Tudi v tem primeru je lokalno iskanje izrazito hitrejše od algoritma CLP.

Pri hiperkockah zaradi dvodelnosti grafa vsi trije algoritmi vračajo enako optimalno množico, ki je seveda enaka ravno polovici moči množice vozlišč. Tudi tu je časovna zahtevnost lokalnega iskanja najmanjša.

Priporočamo uporabo algoritma lokalnega iskanja, saj je hitrejši kot CLP, hkrati pa lahko preprosto dosežemo večjo zanesljivost s povečanjem števila klicev zanke. CLP se je izkazal za uporabnega za preproste analize na manjših grafih. Nasprotno pa bi uporabo relaksacije CLP odvsetovali, saj prepogosto prihaja do optimalne rešitve, ko imajo vsa vozlišča vrednost $\frac{1}{2}$ in nam zato njegov rezultat o največji neodvisni množici da premalo informacij.

## 6. Zahvale

Posebej bi se rad zahvalil asist. dr. Janošu Vidaliju in Žanu Kramarju za nasvete pri pisanju algoritmov in usmeritve glede izvedbe testiranj na grafih.

## LITERATURA

[1] B. S. Baker, *Approximation Algorithms for NP-Complete Problems on Planar Graphs*, Association for Computing Machinery, **41**(1) (1994), 153-180.

[2] *Independent set (graph theory)*, Wikipedia: The Free Encyclopedia, URL: https://en.wikipedia.org/wiki/ Independent_set_(graph_theory) (uporabljeno 1. 2. 2021).

[3] E. W. Weisstein, *Maximum Independent Vertex Set*, MathWorld–A Wolfram Web Resource, URL: https://mathworld.wolfram.com/MaximumIndependentVertexSet.html (uporabljeno 7. 2. 2021).

[4] S. S. Skiena, *The algorithm design manual*, London: Springer, (2012).

[5] A. Hossain, E. Lopez, S. M. Halper, et al. , *Automated design of thousands of nonrepetitive parts for engineering stable genetic systems*, Nature Biotechology, **38** (2020), 1466–1475.

[6] D. Joseph, J. Meidanis, P. Tiwari, *Determining DNA sequence similarity using maximum independent set algorithms for interval graphs*, Berlin, Heidelberg: Springer, v: Algorithm Theory — SWAT '92 (str. 326–337), (1992).

[7] J. Liu, *Maximal and Maximum Independent Sets in Graphs*, Dissertations – 1985, (1992), URL: https://scholarworks.wmich.edu/dissertations/1985 (uporabljeno 24. 2. 2021).

[8] M. Jirri in B. Gartner, *Understanding and using linear programming*, Springer, Edition **1**, (2007).

[9] The Sage Developers, *SageMath, the Sage Mathematics Software System* (Version 9.1.), (2020), URL: https://www.sagemath.org (uporabljeno 10. 1. 2021).

[10] Sagemath, Inc., *CoCalc – Collaborative Calculation and Data Science*, (2021), URL: https://cocalc.com (uporabljeno 10. 1. 2021).

[11] A. E. Brouwer, *The Petersen graph*, URL: https://www.win.tue.nl/ aeb/drg/graphs/Petersen.html (uporabljeno 27. 2. 2021).

[12] D. A. Holton in J. Sheehan, *The Petersen Graph*, Cambridge: Cambridge University Press, Australian Mathematical Society Lecture Series, (1993).

[13] F. Harary, J. P. Hayes in H.-J. Wu, *A survey of the theory of hypercube graphs*, Computers & Mathematics with Applications, **15**(4) (1988), 277–289.