

# Subpath number v kubičnih grafih: minimizacija in protimeri za grafe $L_n$

Žiga Obradović, Lovro Levačić

7. december 2025

## 1 Uvod

V projektu obravnavamo problem minimizacije števila preprostih poti v razredu kubičnih grafov, torej grafov, kjer ima vsako vozlišče stopnjo 3. Naj bo  $G = (V, E)$  povezan graf. *Subpath number* grafa  $G$ , označen s  $pn(G)$ , definiramo kot število vseh preprostih poti v grafu, pri čemer štejemo tudi trivialne poti (zgolj eno vozlišče).

Preprosta pot je zaporedje različnih vozlišč

$$(v_0, v_1, \dots, v_\ell),$$

kjer sta vsaki zaporedni vozlišči povezni z robom grafa. Ker je graf neorientiran, poti, ki gredo v nasprotni smeri, štejemo ločeno.

V literaturi je bila postavljena domneva, da za vsako sodo število vozlišč  $n \geq 10$  obstaja natanko en kubični graf  $L_n$ , ki minimizira subpath number med vsemi kubičnimi grafi na  $n$  vozliščih. Znano je, da domneva za dovolj velika  $n$  ne drži, ni pa znano, pri katerem najmanjšem  $n$  se pojavi prva protimera.

V projektu smo želeli:

- za manjša  $n$  domnevo preveriti izčrpno,
- za večja  $n$  uporabiti metahevrstiko *simulated annealing* in poiskati grafe z manjšim  $pn(G)$  od  $pn(L_n)$ ,
- konstruirati naravne družine kubičnih grafov in jih primerjati z  $L_n$ .

## 2 Teoretično ozadje

### 2.1 Subpath number

Za poljuben povezan graf  $G = (V, E)$  definiramo

$$pn(G) = \#\{\text{preprostih poti v } G\},$$

pri čemer kot poti štejemo tudi trivialne poti dolžine 0.

Število poti zelo hitro raste. Za ilustracijo: en sam rob ima  $pn(G) = 4$ , trikotnik  $pn(G) = 15$ , kvadrat  $pn(G) = 28$ , popoln graf  $K_4$  pa  $pn(G) = 64$ . To nakazuje, da je izračun  $pn(G)$  v splošnem eksponenten v številu vozlišč, zato je natančen izračun smiseln predvsem za grafe z zmernim  $n$  (nekje do približno 20 vozlišč).

## 2.2 Družina grafov $L_n$ in domneva

Grafi  $L_n$  so definirani za soda  $n \geq 10$  in so zgrajeni iz več kopij grafa  $K_4 - e$  (popoln graf na štirih vozliščih, iz katerega odstranimo en rob), ki so povezane v linearno verigo. Na koncih verige so dodani t. i. pendant bloki.

Natančneje:

- če je  $n = 4q - 2$ , je  $L_n$  sestavljen iz  $(n - 10)/4$  kopij grafa  $K_4 - e$ , na obeh koncih pa sta pendant bloka na 5 vozlišč;
- če je  $n = 4q$ , je  $L_n$  sestavljen iz  $(n - 12)/4$  kopij grafa  $K_4 - e$ , na koncih pa sta pendant bloka na 5 in 7 vozlišč.

Geometrijsko gledano dobimo verigo blokov  $K_4 - e$ , ki se na obeh koncih zaključijo z nekoliko večjima komponentama (Slika 1).

Domneva, ki jo preverjamo, je:

**Domneva 10.** V razredu kubičnih grafov na  $n$  vozliščih, kjer je  $n$  soda, je graf  $L_n$  edini graf, ki minimizira subpath number.

Ker je bilo že dokazano, da domneva za dovolj velika  $n$  odpove, nas zanima predvsem, pri katerih najmanjših  $n$  se to zgodi.

## 3 Metodologija

### 3.1 Konstrukcija družin grafov

Osnovni gradnik v vseh naših konstrukcijah je graf  $K_4 - e$ , tj. popolni graf na štirih vozliščih, iz katerega odstranimo en rob. V takem grafu imata dve vozlišči stopnjo 2, preostali dve pa stopnjo 3. Ta dva vozlišča stopnje 2 uporabljamo kot “priklopni točki”, prek katerih gradnike lepimo skupaj.

V kodi (`funkcije2.py`) smo gradnike poimenovali *levi gradnik*, *srednji gradnik* in *desni gradnik* (ter njihove variante z dvema priklopnima vozliščema). Ideja je vedno ista: na  $K_4 - e$  po potrebi dodamo nekaj novih vozlišč, da dobimo pendant blok na 5 ali 7 vozliščih, ki ima eno ali dve priklopni vozlišči. Pomožni funkciji `add_gadget` in `add_gadget2` poskrbita za pravilno oštevilčenje in povezavo gradnikov.

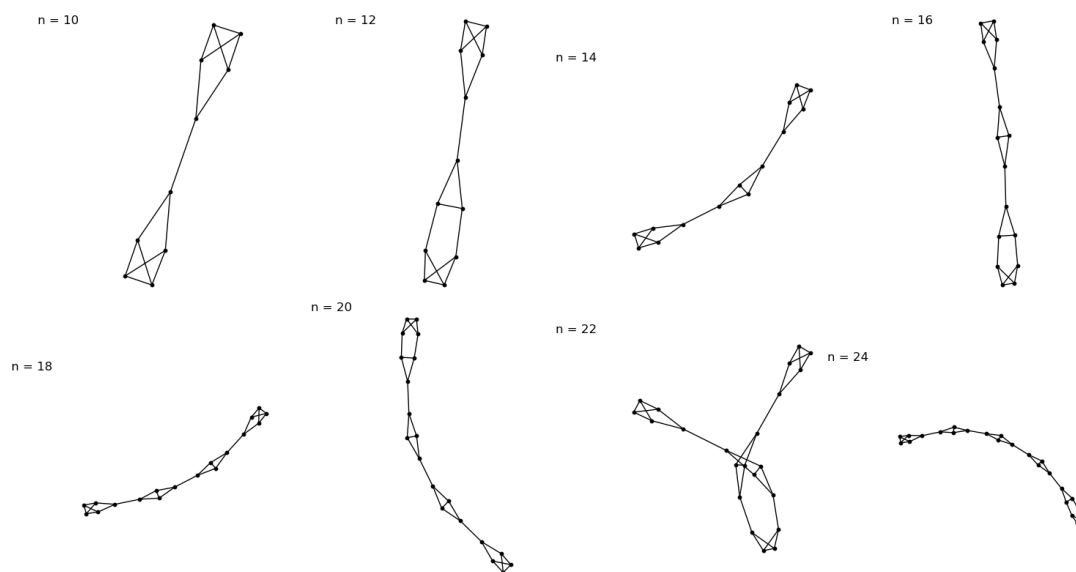
Na tej osnovi definiramo tri družine kubičnih grafov.

**Družina  $L_n$ .** Graf  $L_n$  zgradimo v skladu z definicijo: začnemo z levim gradnikom, dodajamo verigo srednjih gradnikov in verigo zaključimo z levim ali desnim gradnikom tako, da dobimo kubični graf na  $n$  vozliščih. Za različne ostanke  $n$  modulo 4 dobimo različne dolžine verige. Primeri grafov  $L_n$  za  $n = 10, 12, \dots, 24$  so na Sliki 1.

**Goseničasta (zvezdasta) družina.** Druga konstrukcija (`build_caterpillar`) uporablja iste gradnike, vendar jih ne zložimo v eno verigo, temveč dobimo bolj zvezdasto strukturo. Odvisno od  $n \bmod 6$  začnemo z levim ali desnim gradnikom, po potrebi vstavimo še nekaj vmesnih gradnikov, na koncu pa ponovno zaključimo z levim ali desnim gradnikom. Tipičen graf ima nekaj “krakov” z pendant bloki in bolj gosto jedro (Slika 2).

**Drevesna družina  $Tree_n$ .** Tretja družina izhaja iz dejanskega drevesa. Najprej zgradimo drevo  $T_k$  na  $k$  vozliščih: v korenu imamo vozlišče stopnje 3, nato pa postopoma razvejamo liste, dokler ne dosežemo želenega števila vozlišč. Na koncu ima drevo veliko listov stopnje 1.

Funkcija `build_tree` potem na vsak list pripne gradnik tipa  $K_4 - e$  (z dvema priklopnima vozliščema) in ga z obema priklopnima vozliščema poveže na isti list. S tem dvignemo stopnjo



Slika 1: Primeri grafov  $L_n$  za soda  $n$  med 10 in 24.

lista iz 1 na 3, novi vrhovi gradnikov pa so prav tako stopnje 3. Tako dobimo kubični graf na  $n$  vozliščih, ki ga označimo z  $\text{Tree}_n$  (Slika 3).

### 3.2 Izračun subpath number

Za izračun  $pn(G)$  smo napisali funkcijo `subpath_number(G)`. Ideja je naslednja:

- vozlišča oštevilčimo z  $0, \dots, n-1$  in množico obiskanih vozlišč predstavimo z bitno masko;
- zgradimo seznam sosedov;
- za vsako vozlišče  $v$  zaženemo globinsko iskanje (DFS), ki po grafu sprehaja vse preproste poti, ki se začnejo v  $v$ ;
- pri vsakem rekurzivnem klicu pot, ki se konča v trenutnem vozlišču, štejemo kot eno (tako dobimo tudi trivialne poti), nadaljujemo pa samo na sosedo, ki še niso v maski.

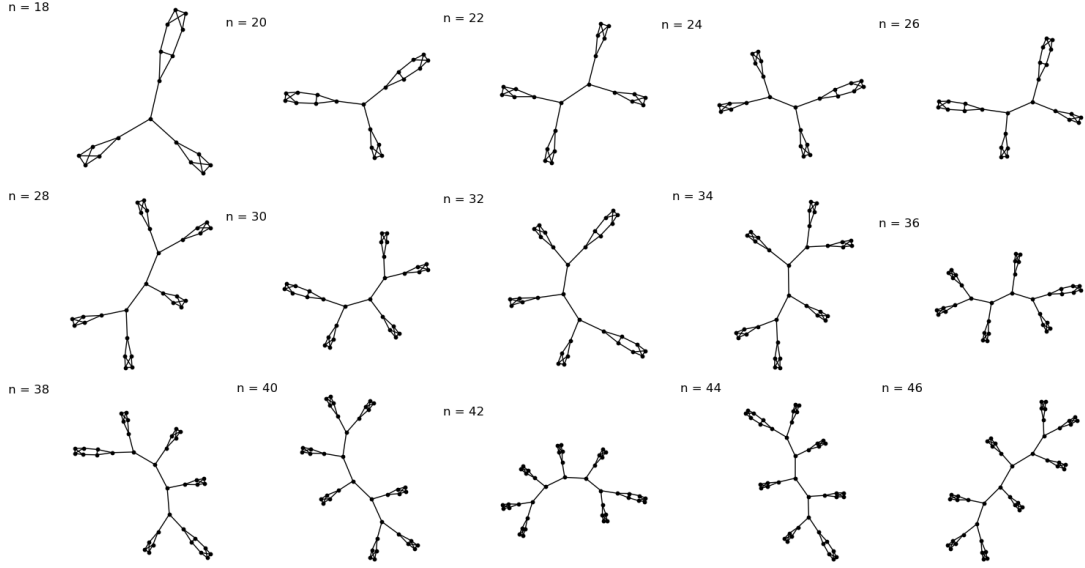
Tako vsako pot preštejemo natanko enkrat. Ker moramo v najslabšem primeru obravnavati ogromno število poti, je časovna zahtevnost eksponentna. V praksi to pomeni, da je izračun še izvedljiv za grafe s približno 20 (oz. kvečjemu 22) vozlišči; za večje grafe ga uporabljamo le v kombinaciji z metahevrstiko in omejenim številom korakov.

### 3.3 Generiranje vseh kubičnih grafov

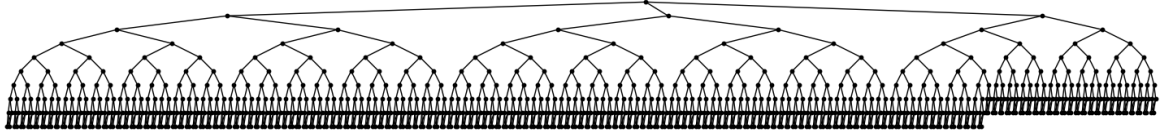
Za manjša  $n$  želimo domnevo preveriti izčrpno, zato potrebujemo generator vseh neizomorfni povezanih kubičnih grafov na  $n$  vozliščih. Uporabimo vgrajeno funkcijo `graphs.nauty_geng` iz SageMath-a (program *geng* iz paketa *nauty*).

V funkciji `cubic_graphs(n)` nastavimo parametre tako, da:

- ima vsak graf natanko  $n$  vozlišč,
- je minimalna in maksimalna stopnja vozlišč enaka 3,
- je graf povezan,
- dobimo po en predstavnik iz vsakega izomorfne razreda.



Slika 2: Goseničasti (zvezdasti) grafi za soda  $n$  med 18 in 46 (konstrukcija `build_caterpillar`).



Slika 3: Primer grafa iz družine  $\text{Tree}_n$  v drevesnem razporedu.

Za soda  $n$  od 4 do 22 smo prešteli, koliko takih grafov dobimo. Rezultati so v Tabeli 1.

Rast je zelo hitra: pri  $n = 18$  imamo že več kot 40 000 grafov, pri  $n = 20$  več kot pol milijona, pri  $n = 22$  pa več milijonov. To je glavni razlog, da smo za  $n \geq 20$  prešli na metahevrstične metode.

### 3.4 Metahevrstika simulated annealing

Za večja  $n$  domneve ne moremo preverjati izčrpno, zato smo uporabili *simulated annealing* (SA). Gre za naključno iskanje, pri katerem na začetku sprejemamo tudi poslabšanja, kasneje pa skoraj samo še izboljšave.

**Sosednji graf.** Na prostoru povezanih kubičnih grafov definiramo sosesko z eno 2-robno zamenjavo (double-edge swap), implementirano v funkciji `random_cubic_neighbor`. Iz trenutnega grafa  $G$  naključno izberemo dva roba  $\{u, v\}$  in  $\{x, y\}$  s štirimi različnimi vozlišči, oba roba odstranimo in ju nadomestimo z eno od dveh novih paritev ( $\{u, x\}, \{v, y\}$  ali  $\{u, y\}, \{v, x\}$ ). Kandidat sprejmemo le, če ne nastanejo zanke ali večrobi in graf ostane povezan ter 3-regularen.

**Energija in temperatura.** Energija stanja je  $E(G) = pn(G)$ . Če ima sosednji graf  $G'$  manjšo ali enako energijo, ga vedno sprejmemo. Če je  $E' > E$ , ga sprejmemo z verjetnostjo

$$\exp\left(-\frac{E' - E}{T}\right),$$

kjer je  $T$  trenutna temperatura. Temperaturo znižujemo po eksponentni shemi  $T_{k+1} = \alpha T_k$  z  $\alpha < 1$ . Parameter  $T_0$  izberemo tako, da na začetku razmeroma pogosto sprejemamo poslabšanja, ob koncu pa skoraj nikoli.

$n$	# kubičnih grafov
4	1
6	2
8	5
10	19
12	85
14	509
16	4060
18	41301
20	510489
22	7319447

Tabela 1: Število neizomorfnih povezanih kubičnih grafov na  $n$  vozliščih.

SA smo v praksi poganjali z nekaj deset tisoč koraki, kot začetni graf pa smo vzeli bodisi  $L_n$  bodisi kak graf iz družine  $\text{Tree}_n$ .

## 4 Eksperimentalni rezultati

### 4.1 Preverjanje domneve za manjša $n$

Za soda  $n$  do 18 smo s funkcijo `cubic_graphs(n)` pridobili vse neizomorfne povezane kubične grafe na  $n$  vozliščih in za vsak graf  $G$  izračunali  $pn(G)$ . Za vsak  $n$  smo posebej primerjali  $pn(G)$  z  $pn(L_n)$ .

Za  $n = 16$  in  $n = 18$  smo izračun izvedli v celoti. Pri  $n = 16$  smo obdelali vseh 4060 grafov, pri  $n = 18$  pa več kot 40 tisoč grafov. V obeh primerih smo našli več kubičnih grafov z manjšim subpath number kot pri  $L_n$ . To pomeni, da  $L_{16}$  in  $L_{18}$  nista minimizatorja, zato domneva v tej obliki že pri teh vrednostih odpove.

### 4.2 Simulated annealing za večja $n$

Za  $n \geq 20$  izčrpen pregled ni več smiseln, zato smo uporabili SA. Algoritem smo zagnali za več sodih  $n$  med 20 in 32, in sicer z različnimi začetnimi grafi (grafi  $L_n$  in grafi  $\text{Tree}_n$ ).

Pri nekaterih manjših  $n$  (npr.  $n = 10, 12, 14$ ) se je pokazalo, da SA iz  $L_n$  ne najde boljšega grafa; energija se v daljših tekih praktično ne spremeni. To je skladno z intuicijo, da so grafi  $L_n$  pri majhnih  $n$  vsaj lokalno dobri.

Za večja  $n$  smo dobili bolj razgibano sliko. Pri nekaterih parametrih je SA precej znižal energijo v prvih fazah in našel grafe z manjšim  $pn(G)$  od izbranega začetnega. Kot zanimiv primer izpostavimo  $n = 32$ , kjer smo začeli v grafu  $\text{Tree}_{32}$ . V daljšem teku (okoli 20 tisoč korakov) se energija ni znižala, kar kaže, da je  $\text{Tree}_{32}$  v izbrani soseski zelo dober lokalni minimum. Podobno obnašanje smo opazili tudi pri nekaterih drugih drevesnih konstrukcijah.

Zaradi časovne zahtevnosti izračuna  $pn(G)$  pri večjih  $n$  rezultatov SA za zdaj razumemo predvsem kot indikativne: dajo nam kandidate za dobre grafe, ne pa nujno globalnih minimizatorjev.

## 5 Zaključek

V poročilu smo opisali, kako smo pristopili k problemu minimizacije subpath number v razredu kubičnih grafov. Najprej smo implementirali natančen izračun  $pn(G)$  in generator vseh kubičnih grafov na danem številu vozlišč. Na tej osnovi smo:

- prešteli vse neizomorfne povezane kubične grafe za soda  $n$  do 22,
- za  $n = 16$  in  $n = 18$  izčrpno primerjali  $L_n$  z vsemi kubičnimi grafi in našli več grafov z manjšim subpath number,
- konstruirali dve dodatni družini kubičnih grafov (goseničasto družino in drevesno družino  $\text{Tree}_n$ ),
- za večja  $n$  uporabili simulated annealing in preverili, ali lahko iz danega začetnega grafa pridemo do grafa z manjšim  $pn(G)$ .

Glavno sporočilo je, da grafi  $L_n$  že pri razmeroma majhnih  $n$  niso minimizatorji subpath number med vsemi kubičnimi grafi. Za večja  $n$  se kombinacija konstrukcijskih družin in SA izkaže za uporaben pristop: omogoča nam, da vsaj približno raziskujemo prostor kubičnih grafov, čeprav natančen izračun  $pn(G)$  ostaja ozko grlo.

V nadaljevanju bi bilo zanimivo poskusiti še z drugimi metahevrstikami (npr. tabu iskanjem ali genetskimi algoritmi) in hkrati poiskati bolj pametne aproksimacije za  $pn(G)$ , da bi lahko obravnavali tudi grafe z bistveno več vozlišči.