UNIVERZA V LJUBLJANI
FAKULTETA ZA MATEMATIKO IN FIZIKO

Finančna matematika – 1. stopnja

Luka Lodrant

Razbitje grafa na dva disjunktna liha podgrafa

Projekt v povezavi z OR

Ljubljana, 2019

Kazalo

1. Navodilo

In the paper https://arxiv.org/abs/1802.07991 the authors gave a characterization when a graph can be decomposed into two disjoint odd subgraphs. Their result, based on the Gaussian elimination method, implies a polynomial time algorithm to verify this kind od decomposability. Implement this algorithm; then for small values of n, verify the percentage of graphs on n vertices that can be decomposed in the prescribed way. Similarly, you can also consider other classes of graphs, say small bipartite graphs, trees, ...

2. Uvod

Osnovni predmet obravnave tega dela bodo *multigrafi*, ki lahko vzporedne povezave, nimajo pa zank. Grafu brez podvojenih povezav in zank rečemo *enostaven graf*. Naj bo $G = (V(G), E(G))$ multigraf. Z $OddV(G)$ bomo označili množico oglišč z lihimi stopnjami, z $EvenV(G)$ pa množico oglišč s sodimi stopnjami. Lih (oz. sod) podgraf grafa G je podgraf, katerega vsa oglišča so lihe (oz. sode) stopnje.

Naš cilj je, da ugotovimo, kdaj je možno graf razbiti na 2 podgrafa lihe stopnje in za to implementirati polinomski algoritem. Z uporabo tega algoritma pa bomo v drugem delu analizirali kakšen delež posebnih kategorij grafov je možno na tak način razbiti.

Projekt bomo implementirali v programskem jeziku *Python 3*, veliko pa bomo uporabljali knjižnico *networkx* za delo z grafi.

3. Matematična podlaga

Kot prej, naj bo G multigraf z oglišči $V(G)$ in povezavami $E(G)$. Z $EvenV(G)$ bomo označili oglišča sode stopnje, z $OddV(G)$ pa oglišča lihe stopnje. Za množico oglišč U grafa G, naj bo $\langle U \rangle_G$ podgraf induciran na teh ogliščih. Red grafa označimo z $|G|$ in pomeni število oglišč v grafu.

Definicija 3.1. Graf G je *lih*, če je stopnja vsakega oglišča grafa liha.

Definicija 3.2. *Liho razbitje grafa* G dobimo tako, da je tako razbitje povezav grafa na 2 disjunktni množici U in V, da vsaka inducira lih podgraf.

Definicija 3.3. Imamo graf $G = (V, E)$ in množico oglišč $T \subseteq V$. *T-join* je taka podmnožica povezav $J \subseteq E$, da imajo v grafu (V, J) vsa oglišča iz T liho stopnjo.

Za graf G nas torej zanima ali liho razbitje obstaja in kakšno je. Tu bomo opisali polinomski algoritem za ta izračun, dokaz in ideja tega algoritma pa sta na voljo v

Naj bo $\mathcal{X}$ množica povezanih komponent grafa $\langle OddV(G) \rangle_G$. Množico povezanih komponent grafa $\langle EvenV(G) \rangle_G$ razbijemo, množica $\mathcal{Y}$ naj vsebuje komponente lihega reda, $\mathcal{Z}$ pa sodega. Z $\mathrm{N_G}(U)$ označimo množici tistih $X_i \in \mathcal{X}$, ki so z U povezani z lihim številom povezav. Torej $\mathrm{N_G}(U) = \{\, X_i \in \mathcal{X};\, e_G(U, X_i) \text{ je liho} \,\}$. Za vsak X_i definiramo binarno spremenljivko x_i, ki nam pove, ali bomo oglišča iz X_i pobarvali z rdečo ($\mathcal{R}$) ali modro ($\mathcal{B}$). V primeru $x = 1$, bo $X_i \in \mathcal{R}$, v primeru $x = 0$ pa $X_i \in \mathcal{B}$.

Sedaj sestavimo sistem linearnih enačb v polju GF(2):

$$\sum_{X_i \in \mathrm{N_G}(Y)} x_i = 1 \qquad \text{za } Y \in \mathcal{Y}$$

$$\sum_{X_i \in \mathrm{N_G}(Z)} x_i = 1 \qquad \text{za } Z \in \mathcal{Z}$$

Ta sistem rešimo z uporabo primernih metod, v naši implementaciji smo uporabili klasično Gauss-Jordan eliminacijo. Če sistem ni rešljiv, grafa G ni možno razbiti na 2 liha podgrafa. Sedaj naj bo $x_1, \ldots, x_n$ ena izmed rešitev tega sistema. Iz te rešitve sestavimo množici $\mathcal{R}$ in $\mathcal{B}$, nato pa vse povezave, ki mejijo na katerokoli oglišče iz unije $\mathcal{R}$ pobarvamo z rdečo.

Na tej točki imamo pobarvanih nekaj povezav, od katerih pa nobena ne povezuje množic $\mathcal{R}$ in $\mathcal{B}$, prav tako pa tudi nobena ni del grafa $\langle EvenV(G) \rangle_G$. Z T označimo množico vseh vozlišč grafa G, ki imajo sodo rdečo stopnjo (stopnjo, pri kateri gledamo le rdeče povezave). Preostane nam še izračun *T-join* na tem grafu.

Obravnavali bomo vsako povezano komponento posebej. Naj bodo C_i povezane komponente grafa G in definiramo še $T_i = C_i \cap T$. Vsako T_i razbijemo na pare (u, v) (iz pravilnosti algoritma sledi, da je sodega reda) in najdemo preprosto pot med vozliščema u in v. Preprosta pot vsebuje vsako povezavo le enkrat, zaradi povezanosti mora obstajati. Nato preštejemo, kolikokrat je bila vsaka povezava uporabljena v teh preprostih poteh. V naš *T-join* spadajo tiste, ki so bile uporabljene liho krat.

Sedaj smo z našim algoritmom zaključili, vse povezave iz izračunanega *T-joina* pobarvamo z rdečo, z modro pa pobarvamo vse še nepobarvane povezave. Če je algoritem uspel (linearni sistem je bil rešljiv), modro in rdeče pobarvane povezave inducirajo dva liha podgrafa grafa G.

4. Implementacija algoritma

Kot že povedano smo se implementacije lotili v programskem jeziku *Python 3*, za delo z grafi smo se odločili za uporabo knjižnice *networkx*. Algoritem je implementiran v datotek *decomposition/decomposition.py*.

Pri reševanju linearnega sistema v GF(2) smo si pomagali z *Matrix* razredom [2].

Ko smo enkrat razumeli, kako prej opisan algoritem deluje, njegova implementacija ni bila posebno zahtevna, vse je bolje opisano v komentarjih poleg kode. Paziti je bilo potrebno na par posebnih primerov, kot npr. razbitje že lihega grafa, pri katerem linearni sistem ne obstaja.

5. Analiza posebnih grafov

5.1. **Mali grafi.** *decomposition/small_ graphs.py*

Prva stvar, ki nas je zanimala, je bila kolikšen delež grafov na malem številu oglišč je možno razbiti na dva liha grafa. Grafe smo zgenerirali tako, da smo za vsak n sestavili množico vseh možnih povezav v grafu, potem pa sestavili njeno potenčno množico. Iz te skupine grafov smo nato še odstranili vse grafe z izoliranimi vozlišči,

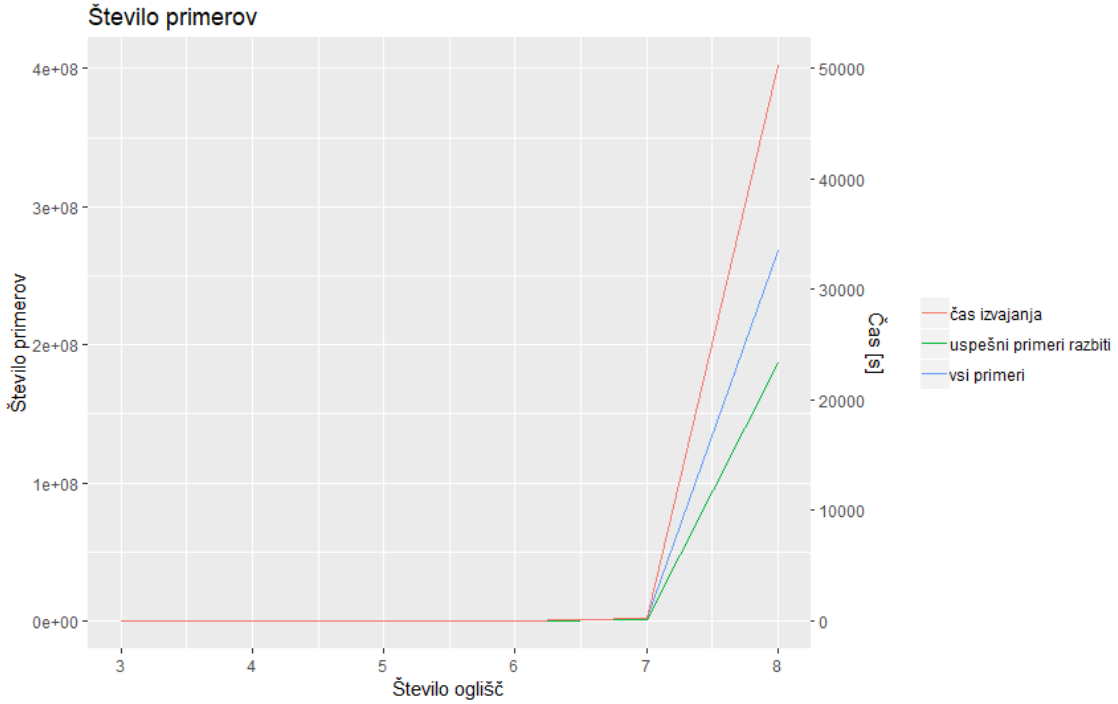
saj so to efektivno grafi na manj vozliščih. Nato smo vsak tako dobljen graf poskusili razbiti in le prešteli, za koliko grafov je bilo to možno.

Do $n = 7$ je tak princip deloval, nato pa smo pri $n = 8$ prišli do težave, saj na toliko vozliščih obstaja kar $268435456 \approx 2.6 * 10^8$ grafov. Z osnovnim algoritmom bi tako potrebovali približno $110h \approx 4,5dneva$, kar pa nismo želeli. Algoritem smo najprej malo optimizirali, namesto sesnamov smo na primernih mestih uporabili sete oz. generatorje, in na začetku preverili ali ima graf kakšno izolirano oglišče, saj takšnega grafa ni mogoče razbiti.

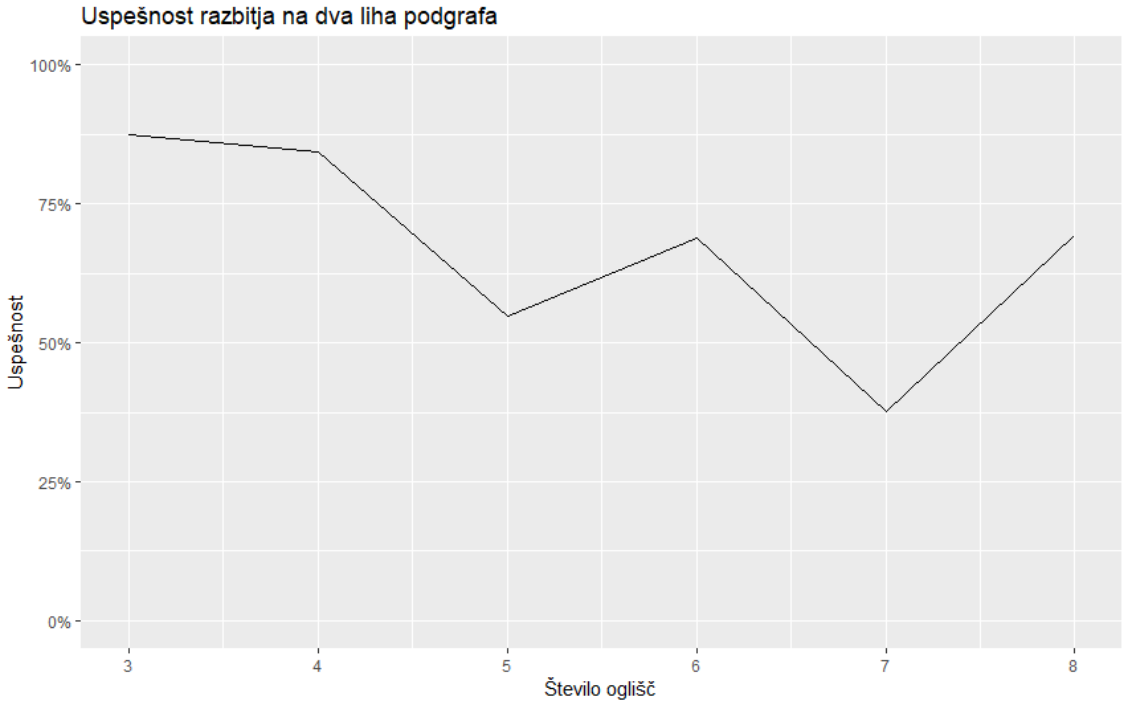
To ni bilo dovolj, ocenjeni čas je zmanjšalo le na približno $70h$. Sedaj bi lahko poskusili algoritem še izboljšati z uporabo nižjenivojskega jezika in bolj primernih podatkovnih struktur, kar pa bi nam lahko vzelo precej več dela. Namesto tega, smo se odločili, da bomo le uporabili malo več procesorske moči. Primerno smo razbili našo testno množico in paralelizirali razbitja z uporabo knjižnice *joblib*. Tako smo na enem samem osemjedrnem računalniku s 14 nitmi prišli do tega, da je izračun trajal "le"$4.5h$.

Rezultate testov predstavimo v tabeli in pa v grafih:

število vozlišč	**število grafov**	**število uspešnih**	**uspešnost**	**čas izvajanja**
3	8	3	37.50 %	0.82 s
4	64	35	54.68 %	0.49 s
5	1024	345	33.69 %	0.66 s
6	32768	19894	60.71 %	3.36 s
7	2097152	638967	30.47 %	144 s
8	186344860	268435456	69.42 %	50278 s



Prvi graf nam za občutek prikaže kako hitra je rast števila grafov, zanima pa nas predvsem drugi.



Tukaj prvič vidimo nihanje uspešnosti med lihimi in sodimi števili vozlišč. Na lihih je uspešnost razbitja manjša kot na sodih. Ta rezultat bomo kasneje videli tudi v drugih primerih grafov. Jasnih trendov na tem grafu ne moremo opaziti, saj imamo premalo podatkov. Tisti pri $n = 3$ in $n = 4$ zaradi malega števila grafov samih niso realen pokazatelj trenda.

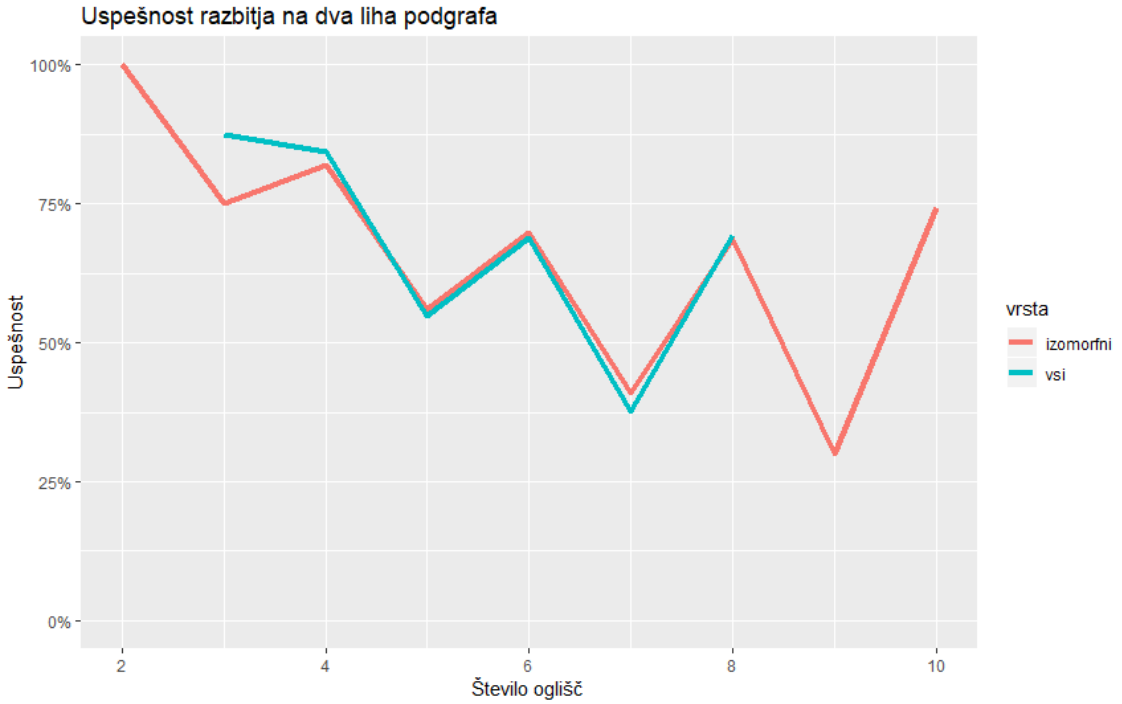
5.2. **Izomorfni razredi grafov.** *decomposition/small_graphs_isomorphic.py*

Tako opazimo, da z analizo vseh možnih grafov ne bomo prišli daleč, saj njihovo število prehitro raste. Malo dlje lahko pridemo tako, da se omejimo le na izomorfne razrede grafov. Teh ne bomo generirali samo, temveč smo jih prenesli iz [4] v obliki *graph6*.

Rezultate spet predstavimo z isto tabelo in grafi.

število vozlišč	**število grafov**	**število uspešnih**	**uspešnost**	**čas izvajanja**
2	2	2	100.00 %	0.61 s
3	4	3	75.00 %	0.39 s
4	11	9	81.81 %	0.38 s
5	34	19	55.88 %	0.36 s
6	156	109	69.87 %	0.09 s
7	1044	428	41.00 %	0.62 s
8	12346	8465	68.56 %	7.66 s
9	274668	82138	29.90 %	50.1 s
10	12005168	8910781	74.22 %	3043 s

Na graf smo dodali tudi uspešnost na vseh in vidimo, da se skoraj ujemata. Bolje lahko prepoznamo tudi vzorec lihih in sodih, kjer je pri lihin n uspešnost nižja kot pri sodih. Opazimo lahko tudi, da za lihe n uspešnost pada, medtem ko za sode počasi raste, vendar ne moremo trditi, da je to pravilo, saj je podatkov premalo.

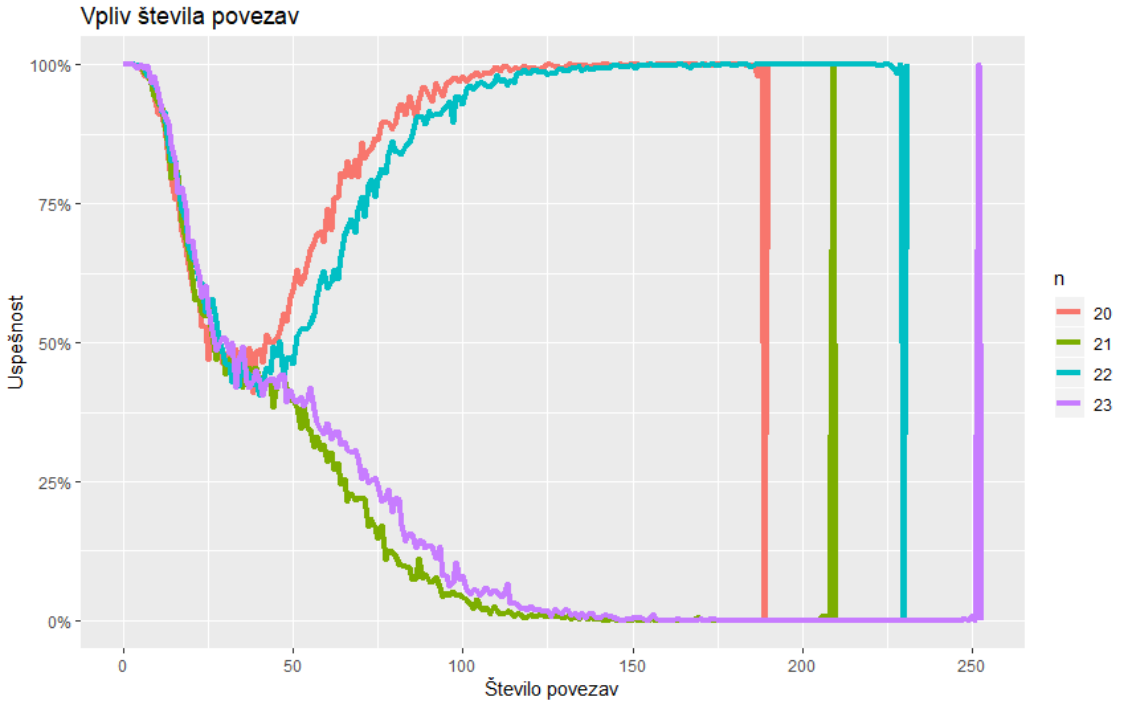


5.3. **Vpliv števila povezav.** *decomposition/random_graphs.py*

Zanima nas, kako število povezav vpliva na razcepnost grafov pri istem številu oglišč. Za ta namen pri konstatnem številu oglišč n generirali po 10000 naključnih grafov z m povezavami za vsak možen m. To bomo storili z uporabo funkcije `nx.gnm_random_graph(n, m)`, ki nam z enako verjetnostjo vrne vsak graf z n oglišči in m povezavami. To bomo storili tudi za več različnih n in poskusili ugotoviti, ali med vsemi temi parametri obstaja kakšna povezava.

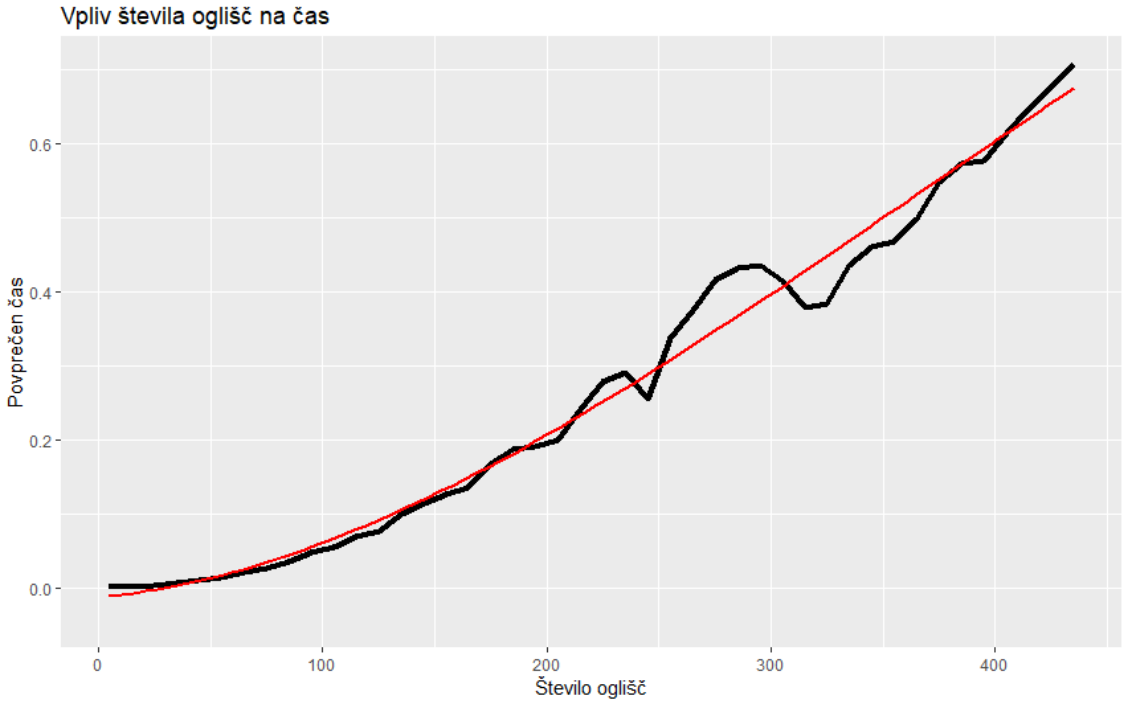
Grafe bomo generirali naključno, saj nas ta vpliv zanima za velike n za katere ne moremo preizkusiti vseh. Izbrali smo si analizo n od 20 do 23. Tabela nam v tem primeru ne pomaga, zato si le izrišemo graf uspešnosti v odvisnosti od števila povezav.

Spet lahko ločimo sode od lihih primerov. Na lihem številu vozlišč verjetnost s številom povezav pada, medtem ko za sode velja, da so grafi z malim in tisti z velikim številom povezav so večinoma razcepni, za tiste vmes pa je verjetnost, da so razcepni nižja. Pri grafu z vsemi povezavami oz. tistemu z eno manj vidimo, da razcepnost skoči, to pa le zato, ker tak graf obstaja le en, in je zato lahko verjetnost le 0 oz. 1.



5.4. **Čas delovanja.** *decomposition/timedifficulty.py*

Za konec lahko preverimo še, kako z rastjo števila oglišč raste čas izvajanja algoritma. Za ta namen bomo grafe spet generirali naključno, ampak drugače kot prej. Namesto fiksnega števila povezav, bomo za vsako povezavo rekli, da jo z verjetnostjo 0.7 dodamo v graf. Nato smo za vsak deseti n poskusili razbiti 100 takšnih grafov in izračunali povprečen čas izvajanja za en primer.



Čase izvajanja prikažemo na grafu in vidimo, da se polinomska krivulja tretje stopnje zelo prilega, kar tudi empirično potrjuje teoretično časovno zahtevnost. Odstopanja nastajajo zaradi majhnega števila naključnih primerov. Če bi to število dvignili, je linija bolj gladka, ampak za izračun porabimo preveč časa. Za naše potrebe je 100 primerov vsake velikosti povsem dovolj.

6. Zaključek

Literatura

[1] M. Kano, G. Y. Katona, K. Varga (2018). Decomposition of a graph into two disjoint. Pridobljeno iz spletne strani: https://arxiv.org/abs/1802.07991.

[2] Project Nayuki (2017). Gauss-Jordan elimination over any field. Pridobljeno iz spletne strani: https://www.nayuki.io/page/gauss-jordan-elimination-over-any-field.

[3] C. Chekuri, B. Raichel (2010). Combinatorial Optimization (Lecture). Pridobljeno iz spletne strani: https://courses.engr.illinois.edu/cs598csc/sp2010/Lectures/Lecture13.pdf.

[4] B. McKay. Graphs (spletni vir). Pridobljeno iz spletne strani: http://users.cecs.anu.edu.au/~bdm/data/graphs.html.