Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Kristjan Reba

Izboljšave dinamičnega algoritma za iskanje maksimalne klike v proteinskem grafu z uporabo strojnega učenja

magistrsko delo

magistrski program druge stopnje
računalništvo in informatika

Mentor: doc. dr. Matej Guid
Somentor: izr. prof. dr. Janez Konc

Ljubljana, 2021

Zahvala

Zahvaljujem se mentorju doc. dr. Mateju Guidu za pomoč in mentorstvo. Zahvaljujem se somentorju izr. prof. dr. Janezu Koncu za vodenje in pomoč pri izdelavi magistrskega dela. Zahvaljujem se družini, ker me je podpirala pri študiju, in Evi, ker mi je vedno v oporo.

Kristjan Reba, 2021

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
MCP	maximum clique problem	problem maksimalne klike
SVM	support vector machine	metoda podpornih vektorjev
GNN	graph neural network	graf nevronska mreža
GCN	graph convolutional network	graf konvolucijska mreža
GAT	graph attention network	graf nevronska mreža s pozornostjo
GIN	graph izomorphism network	graf izomorfna mreža

Povzetek

Naslov: Izboljšave dinamičnega algoritma za iskanje maksimalne klike v proteinskem grafu z uporabo strojnega učenja

Iskanje maksimalne klike spada med dobro raziskane NP-polne probleme. Za praktično uporabnost algoritmov za iskanje maksimalne klike morajo biti ti dovolj hitri na ciljni domeni grafov. V zadnjih letih je bilo narejenega veliko napredka na področju strojnega učenja na grafih. V magistrskem delu uporabimo moderne pristope strojnega učenja na grafih za pohitritev dinamičnega algoritma za iskanje maksimalne klike. Pohitritve testiramo na različnih vrstah grafov s poudarkom na različnih vrstah proteinskih grafov. Ugotovimo, da so pohitritve možne in jih lahko dosežemo z dobro izbiro modela za strojno učenje. Ugotovimo tudi, da pohitritve niso velike, vendar pa so konsistentne na skoraj vseh predstavljenih grafih.

Ključne besede

proteinski graf, maksimalna klika, strojno učenje

Abstract

Title: Improvements to the dynamic algorithm for finding maximum clique in a protein graph using machine learning

Finding maximum clique is a well-researched NP-complete problem. For the practical applicability of algorithms for finding the maximum clique, they must be fast enough on the target domain of graphs. There has been a lot of progress made in recent years in the field of machine learning on graphs. In the master's thesis we use modern approaches to machine learning on graphs to speed up the dynamic algorithm for finding the maximum clique. Speedups are tested with different types of graphs with an emphasis on different types of protein graphs. We find that speeding up the maximum clique search is possible and can be achieved with a good choice of machine learning model. We also find that the speedups are not large but are consistent on almost all the graphs presented.

Keywords

protein graph, maximum clique, machine learning

Poglavje 1

Uvod

Iskanje maksimalne klike v grafu je dobro raziskan NP-poln problem [1]. Hitri algoritmi za iskanje maksimalne klike se uporabljajo na področjih, kot so bioinformatika, računska kemija in analiza socialnih omrežij [2, 3]. Iskanje maksimalne klike na področju bioinformatike je ključno pri odkrivanju in razvoju novih zdravil. Pogosto si namreč želimo najti podobnosti med molekulami, kot so na primer proteini. Molekule lahko opišemo kot grafe, kjer vozlišča predstavljajo atome oziroma skupino atomov, povezave pa predstavljajo kemijske vezi. Proteini, katerih struktura je podobna, imajo pogosto podobno funkcionalnost in obnašanje, zato iščemo kar se da optimalno poravnavo atomov oziroma skupin atomov dveh molekul. Iskanje te poravnave lahko izrazimo kot problem iskanja maksimalne klike na produktnem grafu dveh molekul. Preiskovanje velikih podatkovnih baz kemijskih učinkovin in iskanje strukturne podobnosti med učinkovinami in proteini nam omogoča odkrivanje in razvoj novih zdravil.

Za praktično uporabnost algoritma mora ta najti maksimalno kliko v nekem doglednem času. To ni vedno mogoče in različni algoritmi bodo za isti graf potrebovali različno veliko časa. V magistrskem delu se omejimo na izboljševanje dinamičnega algoritma za iskanje maksimalne klike z uporabo metod strojnega učenja. Moderni algoritmi za iskanje maksimalne klike so izredno optimizirani, zato so kakršne koli izboljšave težko dosegljive. Iz tega

razloga v magistrskem delu preverimo potencialne možnosti za pohitritve iskanja maksimalne klike. Veliko novejših algoritmov je usmerjenih v hitro iskanje maksimalne klike na posebnih grafih, kot so na primer ravninski grafi. Želimo si algoritma, ki bi hitro deloval na proteinskih grafih, kar bi lahko pohitrilo iskanje primernih učinkovin in razvoj novih zdravil. V magistrskem delu raziščemo, ali so pohitritve na proteinskih grafih izvedljive in v kolikšni meri, ter poskusimo izboljšati enega izmed hitrejših algoritmov za iskanje maksimalne klike v grafu.

Algoritem, ki ga želimo izboljšati, temelji na principu razveji in omeji ter za učinkovito iskanje maksimalne klike uporablja dinamično prilagajanje glede na potek algoritma [4]. To dinamično prilagajanje določa parameter, ki se ga nastavi pred potekom algoritma. Dobro izbrana vrednost parametra nam lahko v določenih skupinah grafov zelo pohitri iskanje maksimalne klike [4].

V poglavju *Pregled področja* opišemo problem iskanja maksimalne klike in predstavimo algoritme, ki se uporabljajo za hitro iskanje. Predstavimo tudi algoritem MCQD, na katerem kasneje gradimo. V poglavju *Strojno učenje na grafih* predstavimo pristope za strojno učenje na grafih. Opišemo klasično formulacijo problema na grafu in opišemo modernejše pristope, kot so graf nevronske mreže in metoda podpornih vektorjev s posebno vrsto jedrne funkcije. V poglavju *Podatkovne množice* predstavimo različne vrste grafov, ki jih kasneje uporabljamo. Predstavimo naključne grafe, proteinske produktne grafe, majhne proteinske produktne grafe, proteinske sidrne grafe in grafe DIMACS. V poglavju *Metodologija* opišemo pripravo podatkovnih množic, učenje modelov in evalvacijo modelov v primerjavi s privzetim algoritmom. V poglavju *Eksperimenti in rezultati* primerjamo prvoten algoritem MCQD in z različnimi modeli strojnega učenja razširjene algoritme MCQD. V *Zaključku* povzamemo ugotovitve in predlagamo nadaljnje korake za izboljšavo predstavljenih algoritmov.

Poglavje 2

Pregled področja

Reševanje NP-polnih problemov je še vedno zelo aktualno področje računalniške znanosti [5, 6, 7]. Med te probleme spada tudi problem maksimalne klike, za katerega obstaja veliko relativno učinkovitih algoritmov [8, 9, 4, 1]. V tem poglavju najprej predstavimo problem maksimalne klike ter enostaven algoritem za iskanje maksimalne klike v grafu. Nato predstavimo enega izmed bolj učinkovitih algoritmov za iskanje maksimalne klike, ki ga želimo kasneje izboljšati.

2.1 Iskanje maksimalne klike

Naj bo $G = (V, E)$ neusmerjen graf. Naj bo $V = 1, ..., n$ množica vozlišč in $E \subseteq V \times V$ množica povezav. Klika C v grafu G je definirana kot podmnožica množice vozlišč V tako, da med vsakima dvema vozliščema v C obstaja povezava. Klika je maksimalna, ko je njena kardinalnost $|C|$ največja izmed klik v grafu G. Problem maksimalne klike (angl. *maximum clique problem*) ali MCP je najti maksimalno kliko danega grafa v splošnem primeru. Številka klike $\omega(G)$ grafa G je število vozlišč v maksimalni kliki grafa G. Problem maksimalne klike je striktno ekvivalenten dobro znanemu problemu kombinatorične optimizacije maksimalne neodvisne množice (angl. *maximum independent set*) ali MIS. Prav tako je striktno ekvivalenten problemu

minimalnega pokritja vozlišč (angl. *minimum vertex cover*) ali MVC. Problem iskanja maksimalne klike v grafu spada med NP polne probleme. Za to množico problemov ne poznamo algoritma, ki najde rešitev v polinomskem času. Večina algoritmov za iskanje maksimalne klike temelji na principu razveji in omeji (angl. *branch and bound*), razlikujejo pa se predvsem v načinu omejevanja preiskovanega prostora in razvejitve.

2.1.1 Preprost algoritem za iskanje maksimalne klike

V nadaljevanju je predstavljen dobro znan algoritem za iskanje maksimalne klike (Algoritem 1) [10]. Algoritem CP uporablja 2 vhodna parametra; množico C (trenutno rešitev) in množico P (vozlišča, ki so kandidati). Algoritem temelji na pristopu razveji in omeji in pregleduje le tiste veje, ki potencialno vsebujejo boljšo rešitev od trenutne. Algoritem deluje rekurzivno in uporablja globalno spremenljivko C^*, ki vsebuje trenutno največjo najdeno kliko in je hkrati tudi spodnja meja za največjo maksimalno kliko v grafu. Omejevanje se izvaja v 3. vrstici algoritma 1 kjer, če velja $|C| + |P| > |C^*|$, potem lahko potencialno najdemo večjo kliko od trenutne tako, da razširimo trenutno kliko z vozliščem iz množice P. To vozlišče izberemo naključno iz množice P. Različni algoritmi uporabljajo različne hevristike za določanje prioritete izbiranja vozlišča $p \in P$. Če pogoj $|C| + |P| > |C^*|$ ne velja, potem končamo preiskovanje trenutne veje, ker vemo, da v trenutni veji ne obstaja klika, večja od trenutno največje najdene klike C^*.

V [1] naredijo empirično primerjavo različnih modernih algoritmov za iskanje maksimalne klike, kjer se izkaže, da na testni množici grafov noben algoritem ni striktno hitrejši od ostalih. V magistrskem delu se omejimo na algoritem MCQD, ki spada med hitrejše algoritme za iskanje maksimalne klike [4].

Algorithm 1 Preprost algoritem za iskanje maksimalne klike C^*

```
1: procedure CP(C, P)
2:     if |C| > |C*| then C* ← C
3:     if |C| + |P| > |C*| then
4:         for each p ∈ P do
5:             P ← P \ p
6:             C' ← P \ C ∪ p
7:             P' ← P ∩ N(p)
8:             CP(C', P')
```

2.2 MCQD

Algoritem MCQD temelji na principu razveji in omeji ter uporablja približno barvanje grafa kot aproksimacijo za zgornjo mejo velikosti klike [4, 8]. Psevdokoda je predstavljena v algoritmu 2.

Algoritem MCQD hrani največjo najdeno kliko v spremenljivki (množici vozlišč) Q_{max} in trenutno kliko v spremenljivki Q. Kot vhodni parameter funkcija sprejme množico urejenih vozlišč $R \in V(G)$, množico barv in nivo algoritma (level). Vozlišča v množici R so urejena glede na barvo. Nivo (level) nam pove trenutno globino rekurzivne funkcije. V 5. vrstici algoritma MCQD izberemo vozlišče p z najvišjo barvo, ki je tudi zadnje vozlišče v množici $R \in V(G)$, in ga odstranimo iz množice R. V vrstici 7 preverimo, ali obstaja potencialno večja klika od trenutne (podobno kot v algoritmu CP 1). Če takšna klika ne more obstajati, potem izberemo novo vozlišče iz R. V nasprotnem primeru dodamo vozlišče p v trenutno kliko Q (vrstica 8) in v množico R_p shranimo vse njegove sosede, ki so hkrati v množici R (vrstica 9). Če je množica R_p prazna in je $|Q| > |Q_{max}|$, potem nastavimo Q_{max} enako Q. V primeru, da množica R_p ni prazna, izvedemo ponovno barvanje vozlišč v R_p ter rekurzivno kličemo algoritem MCQD. Algoritem po vrnitvi iz rekurzije odstrani vozlišče p iz klike Q in izbere novo vozlišče p ter se ponavlja, dokler množica R ni prazna (vrstica 4).

Število korakov, ki jih potrebuje algoritem MCQD, je zmanjšano tako, da uredimo vozlišča v množici R glede na njihovo stopnjo (število sosedov). Vozlišča uredimo le kadar je množica R dovolj velika. V prvih nivojih preiskovalnega drevesa vemo, da je množica vozlišč R velika. Ko je v množici R veliko kandidatov, nas stane računanje tesnejše zgornje meje velikosti klike precej manj kot nas stane preiskovanje vej, ki ne vsebujejo optimalne rešitve. To ne velja za majhne množice potencialnih vozlišč, ker tesnejše meje niso tako učinkovite v zmanjševanju preiskovanih vej in s tem računske zahtevnosti.

Globina, do katere se izplača razvrščati in računati stopnje vozlišč, je določena dinamično med iskanjem maksimalne klike. V gostih grafih so maksimalne klike po navadi večje kot v redkejših grafih iste velikosti. Pričakovati je, da bo globina, do katere se izplača računati tesnejše meje, na globokih grafih večja kot v redkih grafih. Algoritem uporablja globalni spremenljivki $S[level]$ in $S_{old}[level]$, ki hranita vsoto korakov, ki jih je algoritem MCQD opravil do trenutne globine in prejšnje globine. V algoritmu se pojavi spremenljivka $T[level]$, ki hrani delež korakov $T[level] = S[level]/ALL\_STEPS$ do trenutne globine med vsemi do zdaj opravljenimi koraki (ALL_STEPS). S parametrom T_{limit} lahko omejimo uporabo računanja tesnejših mej samo na določene globine. Dokler velja $T[level] < T_{limit}$, izračunamo stopnje vozlišč, jih razvrstimo in izvedemo približno barvanje z algoritmom $ColorSort$ (Algoritem 3), kjer barvamo vozlišča v R, urejena po njihovih stopnjah. Ko ne velja $T[level] < T_{limit}$, ne izvedemo dodatnih operacij.

Algoritem MCQD uporablja empirično določeno vrednost parametra T_{limit} za dinamično prilagajanje iskanja, ki vpliva na časovno zahtevnost algoritma [4]. Vrednost parametra T_{limit} se nahaja na intervalu med 0 in 1. Če je vrednost nastavljena na 0, potem algoritem nikoli ne računa tesnejše meje. V primeru, da je parameter nastavljen na 1, se na vsakem koraku izračuna tesnejše meje. Avtorji algoritma pokažejo, da empirično določena vrednost parametra ni ustrezna za vse vrste grafov.

Algoritma za približno barvanje vozlišč $ColorSort$ (Algoritem 3) ne bomo

podrobneje predstavljali, saj njegovo razumevanje ni ključno za namene magistrskega dela. Vseeno pa prilagamo psevdo kodo algoritma *ColorSort* (Algoritem 3).

Algorithm 2 Dinamičen algoritem za iskanje maksimalne klike

```
1: procedure MaxCliqueDyn(R,C,level)
2:     S[level] ← S[level] + S[level − 1] − S_old[level]
3:     S_old[level] ← S[level − 1]
4:     while R ≠ ∅ do
5:         choose a vertex p with maximum C(p) (last vertex) from R
6:         R ← R \ {p}
7:         if |Q| + C[index_of_p_in_R] > |Q_max| then
8:             Q ← Q ∪ {p}
9:             if R ∩ Γ(p) ≠ ∅ then
10:                if S[level]/ALL_STEPS < T_limit then
11:                    calculate the degrees of vertices in G(R ∩ Γ(p))
12:                    sort vertices in R∩Γ(p) in descending order with respect
   to their degrees
13:                ColorSort(R ∩ Γ(p), C′)
14:                S[level] ← S[level] + 1
15:                ALL_STEPS ← ALL_STEPS + 1
16:                MaxCliqueDyn(R ∩ Γ(p), C′, level + 1)
17:            else if |Q| > |Q_max| then
18:                Q_max ← Q
19:            Q ← Q \ {p}
```

2.3 Ostali algoritmi za iskanje maksimalne klike

Algoritme za iskanje maksimalne klike lahko razdelimo na eksaktne in aproksimacijske. Eksaktni algoritmi vedno najdejo optimalno rešitev, vendar za to porabijo veliko časa, medtem ko aproksimacijski algoritmi hitro najdejo

približno rešitev, vendar ne nujno optimalno. Algoritma MCQD in CP spadata med eksaktne algoritme. Pogosto se pojavljajo algoritmi, ki so zelo hitri na grafih s posebno strukturo [11, 12]. Takšni algoritmi pogosto dosegajo polinomske čase na posebnih grafih, vendar so izredno počasni na grafih, ki nimajo posebne strukture.

V zadnjih letih se je pojavilo kar nekaj aproksimacijskih algoritmov za reševanje NP polnih problemov, ki temeljijo na strojnem učenju [13]. Takšni algoritmi pogosto nadomestijo ročno narejene hevristike s komponento strojnega učenja, ki se iz velikih podatkovnih množic nauči primerne hevristike.

Primer zelo hitrega algoritma, ki pa nam ne daje nobenega zagotovila, da bo najdena klika maksimalna, je algoritem CombOpt Zero [14]. Avtorji [14] pokažejo, da se graf nevronske mreže lahko naučijo dovolj dobre reprezentacije za iskanje maksimalne klike na grafih različnih velikosti. Učenje graf nevronske mreže postavijo v ogrodje spodbujevalnega učenja, kjer z Monte-Carlo drevesnim preiskovanjem poiščejo maksimalno kliko.

Algorithm 3 Algoritem za barvanje vozlišč

```
1:  procedure ColorSort(R,C)
2:      maxno = 1
3:      k_min = |Q_max| − |Q| + 1
4:      if k_min ≤ 0 then
5:          k_min = 1
6:      j = 0
7:      C_1 = ∅
8:      C_2 = ∅
9:      for i = 0 to |R| − 1 do
10:         p = R[i]
11:         k = 1
12:         while C_k ∩ γ(p) ≠ ∅ do
13:             k = k + 1
14:         if k > maxno then
15:             maxno = k
16:             C_maxno = ∅
17:         C_k = C_k ∪ {p}
18:         if k < k_min then
19:             R[j] = R[i]
20:             j = j + 1
21:     C[j − 1] = 0
22:     for k = k_min to maxno do
23:         for i = 1 to |C_k| do
24:             R[j] = C_k[i]
25:             C[j] = k
26:             j = j + 1
```

Poglavje 3

Strojno učenje na grafih

Veliko praktičnih problemov lahko predstavimo v obliki problema na grafu, ki lahko vsebuje veliko relacijskih informacij med elementi. Vendar pa je delo z grafi precej zahtevno, saj se grafi razlikujejo tako po velikosti kot tudi po topologiji. Če si želimo poenostaviti problem klasifikacije oziroma regresije grafa, lahko uporabimo ekstrakcijo značilk, ki opisujejo graf. Primeri takšnih značilk so: število vozlišč, število povezav, povprečna stopnja vozlišč in mnogo drugih. Problem klasifikacije grafa lahko formuliramo kot klasičen problem nadzorovanega učenja, ko na množici značilk poženemo poljuben algoritem za strojno učenje in napovedujemo razred. Primer algoritma, ki deluje na takšnih podatkih, je gradientno strmensko učenje z drevesi (angl. *gradient tree boosting*), ki ga predstavimo spodaj.

Slabost klasifikacije značilk je izguba informacije o topologiji grafa. Če se temu želimo izogniti, moramo uporabiti algoritme strojnega učenja, ki delujejo neposredno na grafu. Primer takšnih algoritmov so različne graf nevronske mreže (angl. *graph neural networks*) ali GNN in metoda podpornih vektorjev (angl. *support vector machine*) ali SVM s prilagojenimi jedrnimi funkcijami [15, 16]. Tako SVM kot tudi GNN predstavimo v tem poglavju.

3.1 Gradientno strmensko učenje z drevesi

Algoritmi, ki temeljijo na principu gradientnega strmenskega učenja z drevesi so se v zadnjih letih izkazali kot izredno močno orodje za reševanje različnih vrst regresijskih in klasifikacijskih problemov. Izkaže se, da so izredno dobri v klasični formulaciji problema nadzorovanega učenja, ko imamo na voljo množico vrednosti značilk v tabelarični obliki in napovedujemo neko regresijsko vrednost oziroma razred.

Takšno učenje temelji na množici šibkih modelov, kot so na primer majhna odločitvena drevesa [17]. Vsak posamezen šibek model nima dovolj reprezentacijske moči za dobro modeliranje podatkovne množice. Modele strojnega učenja lahko kombiniramo v en boljši model, ki združuje vse manjše in pogosto šibkejše modele v ansamble. Strmensko učenje je način, kako združiti šibkejše modele v ansamble. Strmensko učenje deluje tako, da najprej učimo prvi šibki model na učni množici. Potem začnemo učiti drugi šibki model tako, da se osredotočimo na natančno napovedovanje primerov iz učne množice, na katerih prvi model ne deluje dovolj dobro. Pričakovati je, da bo kombinacija teh dveh modelov boljša kot posamičen šibek model. Proces lahko ponavljamo za poljubno število šibkih modelov. Vsak naslednji model poskuša popraviti pomanjkljivosti prejšnjih modelov.

Gradientno strmensko učenje je verzija strmenskega učenja (angl. *boosting*), ki poskuša minimizirati napako za celotno učno množico. To izvede tako, da kot ciljno vrednost za vsak primer iz učne množice spremeni glede na to, koliko ta primer doprinese k napaki.

Algoritem XGBoost temelji na predstavljenih principih in razširi gradientno strmensko učenje s paralelizacijo, gradnjo odločitvenih dreves in specifično funkcijo napake. Algoritem je znan po tem, da je zelo hiter in daje dobre rezultate s privzetim naborom parametrov [18, 17]. Algoritem XGBoost uporabimo tudi v naših eksperimentih v poglavju Eksperimenti in rezultati. Kot vhodne atribute algoritem XGBoost vsebuje naslednje informacije o grafu: število vozlišč, število povezav, gostoto grafa, povprečno stopnjo vozlišča, najvišjo stopnjo vozlišča in najnižjo stopnjo vozlišča.

3.2 SVM in jedrne funkcije

Metoda podpornih vektorjev (support vector machine ali SVM) je algoritem strojnega učenja, ki se uporablja za reševanje regresijskih in klasifikacijskih problemov. SVM slika podatkovne primere v prostor tako, da maksimizira razdaljo med kategorijama [16]. Poleg linearne klasifikacije lahko SVM izvede nelinearno klasifikacijo z uporabo jedrnega trika (angl. *kernel trick*), ki slika podatke v visoko dimenzionalni prostor značilk. SVM lahko prilagodimo tako, da deluje tudi na regresijskih problemih (angl. *support vector regressor*) ali SVR. Ideja jedrnih funkcij je kreiranje prostora značilk v obliki vektorja, podobno kot tako imenovana vreča besed (angl. *bag-of-words*) ali BOW za graf. BOW se navadno uporablja v domeni procesiranja naravnega jezika. Uporablja preprosto štetje besed, ki jih obravnava kot značilke za opisovanje dokumentov. Podobno lahko naredimo za grafe tako, da vozlišča tretiramo kot besede, a je takšna naivna predstavitev grafov nezmožna ločevati med določenimi grafi. Namesto tega lahko uporabimo tako imenovano vrečo stopenj vozlišč (angl. *bag-of-node-degrees*), ki šteje stopnje vozlišč in tako kot BOW kreira vektorsko vložitev za graf.

Namesto vreče stopenj vozlišč lahko uporabimo grafke (angl. *graphlets*), ki nekoliko bolj povzamejo strukturo grafa. Štetje grafkov je računsko zahtevno, saj za štetje $k-$grafkov na grafu velikosti n porabimo $O(n^k)$ časa in zato ni vedno praktično uporabno.

3.2.1 Weisfeiler-Lehmanova jedrna funkcija

Weisfeiler-Lehmanova (WL) jedrna funkcija je bolj učinkovita od štetja grafkov, saj uporablja iterativno izboljševanje slovarja vozlišč. Nanjo lahko gledamo kot na posplošitev vložitve vreče stopenj vozlič. Algoritem, ki ga uporablja Weisfeiler-Lehmanova jedrna funkcija, je imenovan postopno izpopolnjevanje barv (angl. *color refinement*). Če imamo podan graf G in vozlišča V, želimo vsakemu vozlišču določiti barvo po rekurzivni funkciji

$$c^{(k+1)}(v) = HASH(\{c^{(k)}(v), \{c^{(k)}(u)\}_{u \in N(v)}\}),$$

kjer $HASH$ slika različne vhode v različne barve. Po K korakih postopnega izpopolnjevanja barv $c^{(K)}(v)$ povzame strukturo K-soseščine vozlišča v. Po opravljenem postopnem izpopolnjevanju barv Weisfeiler-Lehmanovo jedro prešteje število vozlišč z določeno barvo. Weisfeiler-Lehmanovo jedro je računsko učinkovito, saj je časovna kompleksnost algoritma za postopno izpopolnjevanje barv na vsakem koraku linearna v odvisnosti od števila povezav v grafu. WL je soroden graf nevronskim mrežam, ki jih predstavimo v naslednjem poglavju [19, 20, 21].

3.3 Graf nevronske mreže

V zadnjem času so se na področju strojnega učenja na grafih uveljavile graf nevronske mreže [15, 22, 5, 23]. Graf nevronske mreže (angl. *graph neural networks*) ali GNN so pristop globokega učenja (angl. *deep learning*), ki operira na grafih, v grobem ločimo na dve skupini glede na njihovo aplikacijo. Ena možnost je reševanje problemov na nivoju vozlišč v grafu, kar pomeni, da lahko vsakemu vozlišču priredimo realno vrednost, če gre za regresijo, oziroma vozlišču priredimo razred, v katerega spada, če gre za klasifikacijo.

Včasih želimo celotnemu grafu prirediti eno samo vrednost, pa naj gre za klasifikacijo ali pa regresijo. Takrat moramo uporabiti takšen GNN, ki je sposoben združiti informacijo iz vozlišč. Primer takega problema bi bilo napovedovanje biološke uporabnosti učinkovin iz grafa molekule, kjer napovedujemo le eno realno vrednost.

Naprej lahko ločimo metode strojnega učenja na nadzorovane in nenadzorovane. Ideje, ki se uporabljajo za delovanje GNN, izhajajo iz konvolucijskih nevronskih mrež (angl. *Convolutional neural network*) ali CNN, kjer moramo za učinkovito učenje uporabljati lokalne operacije, ki si delijo parametre. Razlika med GNN in CNN je predvsem v permutacijski neodvisnosti sosedov, ko imamo opravka z grafi. Z drugimi besedami: zaporedje sosedov nekega vozlišča ni pomembno oziroma niti ni definirano. Graf nevronske mreže za vsako vozlišče naredijo vektorsko vložitev $h_v \in \mathbb{R}^s$ (angl. *embed-*

ding) dimenzije s, ki vsebuje informacijo o sosedih vozlišča v.

Izkaže se, da prvotna oblika grafa nevronskih mrež ne deluje dovolj dobro na določenih problemih, zato so se pojavile različne oblike grafov nevronskih mrež.

3.3.1 Graf konvolucijske nevronske mreže

Graf nevronske mreže, predstavljene v [24], uporabljajo spektralne graf konvolucije. Spektralne konvolucije so definirane v Fourierjevi domeni tako, da izračunamo lastno dekompozicijo (angl. *eigen-decomposition*) Laplaceove matrike grafa. Operacija je definirana kot množenje signala $x \in \mathbb{R}^N$ (skalar za vsako vozlišče) s filtrom $g_\theta = diag(\theta)$, parameteriziran z $\theta \in \mathbb{R}^N$, tako da velja

$$g_\theta \star x = U g_\theta(\Lambda) U^T x,$$

kjer je U matrika lastnih vektorjev normalizirane Laplaceove matrike

$$L = I_N - D^{-\frac{1}{2}} A D^{-\frac{1}{2}} = U \Lambda U^T.$$

D je matrika stopenj vozlišč, A je matrika sosednosti grafa in Λ je diagonalna matrika lastnih vrednosti. Graf konvolucijske nevronske mreže lahko združujemo z različnimi združevalnimi plastmi, kot so združevalni sloj z maksimizacijo (angl. *max pooling*) in združevanje s povprečenjem (angl. *average pooling*). Te plasti nam pomagajo zmanjšati dimenzijo problema in agregirajo informacijo iz več vozlišč.

3.3.2 Graf nevronske mreže s pozornostjo

V zadnjih letih so se na področju procesiranja naravnega jezika uveljavili tako imenovani mehanizmi pozornosti (angl. *attention*) [25]. Mehanizem pozornosti je bil uspešno vpeljan kot računsko učinkovita alternativa rekurzivnih nevronskih mrež in konvolucijskih nevronskih mrež tako na področju obdelave naravnega jezika kot tudi računalniškega vida.

Graf nevronske mreže s pozornostjo (angl. *graph attention networks*) ali GAT vpeljejo mehanizem pozornosti na grafe [26]. Podobno kot GCN, GAT uporablja značilke vozlišča kot tudi sosednjih vozlišč. GAT simulira mehanizem pozornosti tako, da vsakemu vozlišču v soseščini določi utež brez uporabe računsko zahtevnih operacij kot je inverz matrike. Prednosti uporabe mehanizma pozornosti na grafu so hitrost, ki jo pridobimo zaradi lažje paralelizacije, pomembnejša sosednja vozlišča pa dobijo več pozornosti in deluje tako na usmerjenih kot tudi neusmerjenih grafih.

3.3.3 Graf izomorfne nevronske mreže

Grafe lahko primerjamo po njihovi topološki podobnosti. GCN ne more ločiti med nekaterimi strukturami grafov [22]. Graf izomorfne nevronske mreže (angl. *graph isomorphism network*) ali GIN namesto maksimalne in povprečne agregacije vozlišč, kot to počne GCN, uporablja agregacijo seštevanja, ki je bolj robustna glede na strukturo grafa [22]. Čeprav ima GIN večjo reprezentacijsko moč (med seboj loči ne-izomorfne grafe), se v praksi izkaže, da je učenje GIN nekoliko zahtevnejše in ne dosega vedno boljših rezultatov kot GCN [14].

Poglavje 4

Podatkovne množice

Za namene magistrskega dela uporabimo podatkovne množice proteinskih, naključnih in sintetičnih grafov. Generiramo različne vrste proteinskih grafov: majhne produktne grafe, produktne grafe in sidrne grafe (angl. *docking graphs*). Generiramo tudi naključne grafe različnih gostot in velikosti. V tem poglavju predstavimo uporabljene množice grafov.

4.1 Naključni grafi

Naključni graf $G(n, p)$ je generiran tako, da je verjetnost vsake povezave med n vozlišči enaka p. Takšni grafi se imenujejo tudi Erdős–Rényijevi naključni grafi. Med naključne grafe spadajo tudi zelo gosti ($p > 0,99$) in zelo redki grafi ($p < 0,01$).

4.2 Proteinski grafi

Proteine lahko predstavimo kot proteinske grafe. V proteinskih grafih imajo vozlišča prostorske koordinate in so postavljena v funkcionalne skupine površinskih aminokislin. Vozlišča so označena s petimi različnimi barvami, ki predstavljajo pet različnih kemijskih značilnosti. Dve vozlišči proteinskega grafa G sta sosednji, če je razdalja med u_i in u_j manjša od 15 Å. Za iskanje maksi-

malne klike ignoriramo barve vozlišč in njihove prostorske koordinate. Zanima nas le groba struktura molekule. Proteinski grafi nam pomagajo pri modeliranju obnašanja proteinov in poenostavitvi teh velikih molekul.

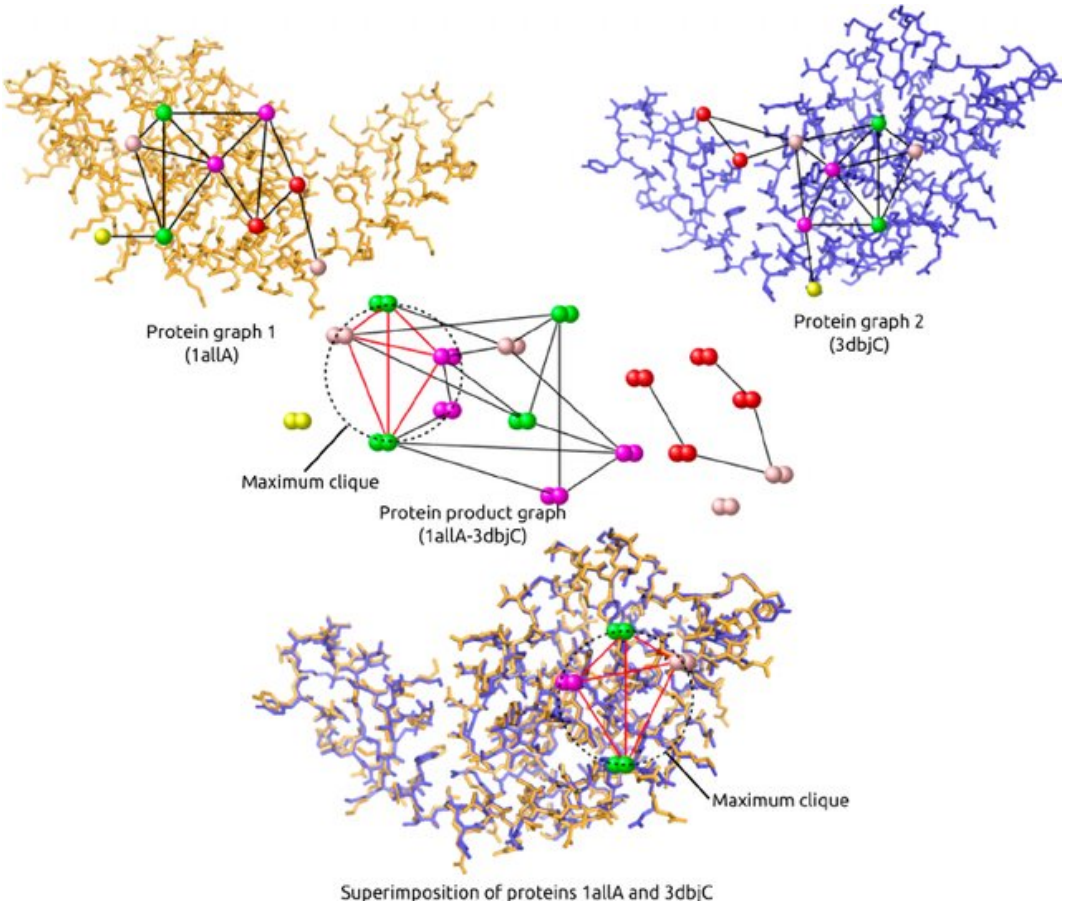
4.3 Proteinski produktni grafi

Dva proteina lahko primerjamo tako, da najdemo maksimalno kliko njunega produktnega grafa. Maksimalna klika je prekrivanje, ki poravna večino vozlišč obeh proteinskih grafov. Proteinski produktni graf dveh proteinskih grafov G_1 in G_2 je definiran na množici vozlišč $V(G_1, G_2) = V(G_1) \times V(G_2)$. Vsako vozlišče v proteinskem produktnem grafu je sestavljeno iz vozlišča iz prvega proteinskega grafa ($u \in G_1$) in vozlišča iz drugega proteinskega grafa ($v \in G_2$). V splošnem ima proteinski produktni graf $|V_1| \times |V_2|$ vozlišč. V praksi zmanjšamo število vozlišč tako, da obdržimo le tista vozlišča, ki imajo podobne sosede v prvotnih grafih. Soseščina vsakega vozlišča v proteinskem grafu je določena kot sfera v premeru 6 Å. Primer proteinskega produktnega grafa je predstavljen na sliki 4.1. Za generiranje proteinskih produktnih grafov uporabimo knjižnico Probis [27]. Problem obdelave velikih grafov je shranjevanje matrike sosednosti vozlišč v pomnilnik. Ker naša implementacija algoritma MCQD uporablja gosto matriko sosednosti in so proteinski produktni grafi izredno veliki in redki, smo se odločili za manjšo učno in testno množico proteinskih produktnih grafov glede na ostale vrste grafov.

4.4 Majhni proteinski produktni grafi

Proteinski produktni grafi so zelo veliki in redki, kar nam otežuje procesiranje takšnih grafov. Iz tega razloga se pogosto uporabljajo majhni proteinski produktni grafi, ki so generirani tako, da razdelimo prvotni proteinski produktni graf na veliko manjših pod-grafov. Dobljeni grafi so relativno majhni in gosti v primerjavi s prvotnim proteinskim produktnim grafom. Prednost majhnih proteinskih produktnih grafov je hitrost, s katero jih lahko procesi-

ramo. Slabost uporabe takšnih grafov je izguba informacije, ki je vsebovana v prvotnem proteinskem produktnem grafu. Primer takšne izgube informacije se pokaže pri iskanju maksimalne klike v proteinskem produktnem grafu, saj če iščemo maksimalno kliko na majhnih pod-grafih, ne moremo jamčiti, da je klika res maksimalna.



Slika 4.1: Primer poravnave dveh grafov. Rezultat poravnave je produktni graf. Slika vzeta iz [2].

4.5 Proteinski sidrni grafi

Proteinski sidrni graf (angl. *docking graph*) je graf, čigar vozlišča so sidrani fragmenti. Sidrani fragmenti so po navadi majhne molekule sidrane v protein v različnih rigidnih pozicijah tako, da imajo kar se da nizko energijo. Dve vozlišči sta med seboj povezani, če sta sidrana fragmenta povezana s povezovalnimi atomi (angl. *linker atoms*) [28, 29].

4.6 Uteženi proteinski sidrni grafi

Uteženi grafi so grafi, ki imajo vsakemu vozlišču $v \in V(G)$ prirejeno vrednost $w \in \mathbb{R}$. Pri uteženih proteinskih sidrnih grafih utež na vozlišču odraža energijo sidranega fragmenta, ki ga predstavlja vozlišče. Energija posameznega vozlišča je izračunana glede na pozicijo sidranega fragmenta v molekuli. Z minimalnimi spremembami lahko algoritem MCQD spremenimo tako, da bo našel maksimalno uteženo kliko v grafu. V praksi nam iskanje utežene maksimalne klike v uteženih proteinskih sidrnih grafih pomaga najti najbolj primeren sidrani fragment za izbrani protein.

4.7 Grafi DIMACS

Množica grafov DIMACS je bila predstavljena kot testna množica sintetičnih grafov za testiranje algoritmov za iskanje maksimalne klike na tekmovanju DIMACS leta 1992 [30]. Grafi so različnih velikosti, topologij in izbrani tako, da so kar se da raznoliki. Za nekatere od teh grafov še vedno ne vemo velikosti maksimalne klike, čeprav število vozlišč ne presega 3300 [30].

Poglavje 5

Metodologija

Primerjati želimo več algoritmov strojnega učenja za izboljšavo algoritma za iskanje maksimalne klike v grafu. V tem poglavju preverimo možnosti za pohitritev algoritma MCQD na različnih vrstah grafov. Zanima nas, za katere množice grafov je privzeta vrednost parametra $T_{limit} = 0,025$ skoraj optimalna. Predstavimo tudi korake, ki jih izvedemo, da lahko kreiramo učno in testno množico ter evalviramo algoritme za strojno učenje.

5.1 Možnosti za izboljšave

Zanima nas, ali je možno pohitriti algoritem MCQD z izbiro potencialno boljše vrednosti parametra T_{limit}. Za vsako podatkovno množico preverimo, kako se čas za iskanje maksimalne klike spreminja z različnimi vrednostmi parametra T_{limit}. Za vsako podatkovno množico naključno vzorčimo tri grafe, za katere preverimo čas izvajanja pri različnih vrednostih parametra T_{limit}, vključno s privzeto vrednostjo $0,025$.

5.1.1 Vpliv začetnega razvrščanja vozlišč

Hitrost iskanja maksimalne klike z algoritmi, ki temeljijo na principu razveji in omeji, je pogosto odvisna od začetnega vrstnega reda preiskovanja vozlišč. Eden takšnih algoritmov je tudi MCQD. V primeru, da že pred iz-

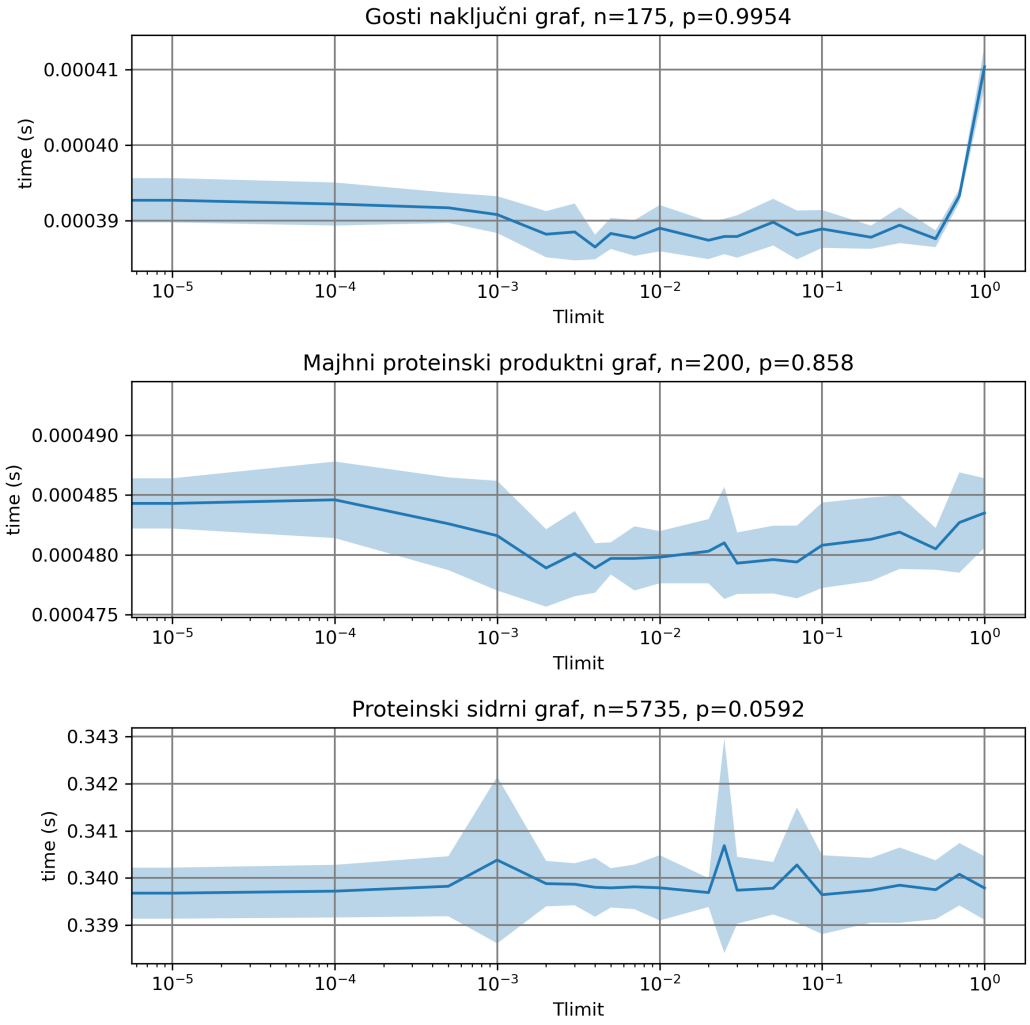
vajanjem uganemo optimalni vrstni red preiskovanja (certifikat NP-polnega problema). potem bi maksimalno kliko našli zelo hitro (v linearnem času). Na množici treh naključnih grafov testiramo vpliv naključnega razvrščanja vozlišč pred potekom algoritma MCQD. Pred začetkom iskanja maksimalne klike naključno pomešamo vrstni red vozlišč. Nato za različne vrednosti parametra T_{limit} poženemo algoritem MCQD in zabeležimo čas. Postopek ponovimo z 20 različnimi začetnimi razvrščanji vozlišč in izračunamo standardno deviacijo časa za vsako vrednost T_{limit}. Rezultati eksperimentov za tri različne grafe so predstavljeni na sliki 5.1. Iz rezultatov lahko ugotovimo, da ima razvrščanje vozlišč majhen vpliv na čas, ki ga algoritem MCQD potrebuje, da najde maksimalno kliko.

V nadaljnjih eksperimentih na posameznem grafu za vse modele uporabljamo enako začetno razvrstitev vozlišč.

5.1.2 Naključni grafi

Za učno množico naključnih grafov narišemo histogram porazdelitve vrednosti optimalnih parametrov T_{limit}. Na sliki 5.2 opazimo, da je porazdelitev vrednosti optimalnega T_{limit} parametra najgostejša med vrednostmi 10^{-3} in 10^{-1}. Iz tega lahko sklepamo, da ne obstaja ena sama vrednost parametra T_{limit}, ki bi bila vedno optimalna za vse grafe iz množice naključnih grafov.

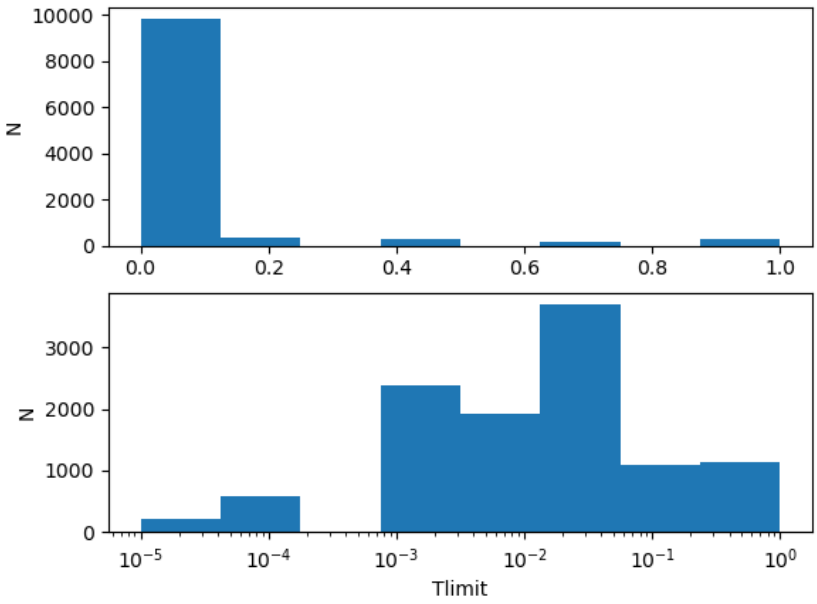
Na sliki 5.3 so predstavljeni časi, ki jih potrebuje algoritem MCQD, da najde maksimalno kliko na naključno izbranem grafu iz učne množice naključnih grafov za različne vrednosti parametra T_{limit}. Rdeča črta označuje privzeto vrednost $T_{limit} = 0,025$. Opazimo, da je privzeta vrednost skoraj optimalna in ne obstaja boljša vrednost parametra T_{limit}, pri kateri bi algoritem MCQD veliko hitreje našel maksimalno kliko na tem specifičnem grafu.



Slika 5.1: Vpliv vrstnega reda preiskovanja vozlišč na različnih grafih je predstavljena na sliki 5.1. Modra črta je povprečen čas za vsako vrednost parametra T_{limit}. Modro obarvano območje predstavlja standardno deviacijo.

5.1.3 Gosti grafi

Na sliki 5.3 so predstavljeni časi, ki jih potrebuje algoritem MCQD, da najde maksimalno kliko na naključno izbranem gostem grafu za različne vrednosti T_{limit} parametra. Rdeča črta označuje privzeto vrednost $0,025$. Opazimo, da

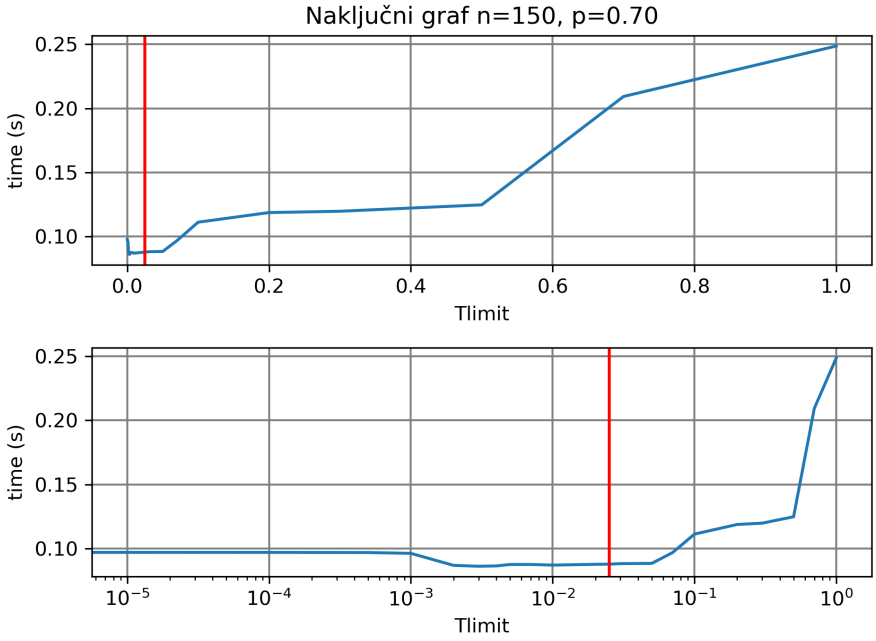


Slika 5.2: Porazdelitev vrednosti T_{limit} za učno množico naključnih grafov na linearni in logaritemski skali.

privzeta vrednost ni optimalna in obstaja boljša vrednost parametra T_{limit}, pri kateri bi algoritem MCQD hitreje našel maksimalno kliko. Pri vrednosti parametra $T_{limit} = 10^{-3}$ bi algoritem MCQD dvakrat prej zaključil z iskanjem maksimalne klike kot pa s privzeto vrednostjo.

5.1.4 Majhni proteinski produktni grafi

Na sliki 5.5 so predstavljeni časi, ki jih potrebuje algoritem MCQD, da najde maksimalno kliko na naključno izbranem majhnem proteinskem grafu za različne vrednosti parametra T_{limit}. Rdeča črta označuje privzeto vrednost $0,025$. Opazimo, da je privzeta vrednost skoraj optimalna in ne obstaja boljša vrednost parametra T_{limit}, pri kateri bi algoritem MCQD veliko hitreje našel maksimalno kliko.



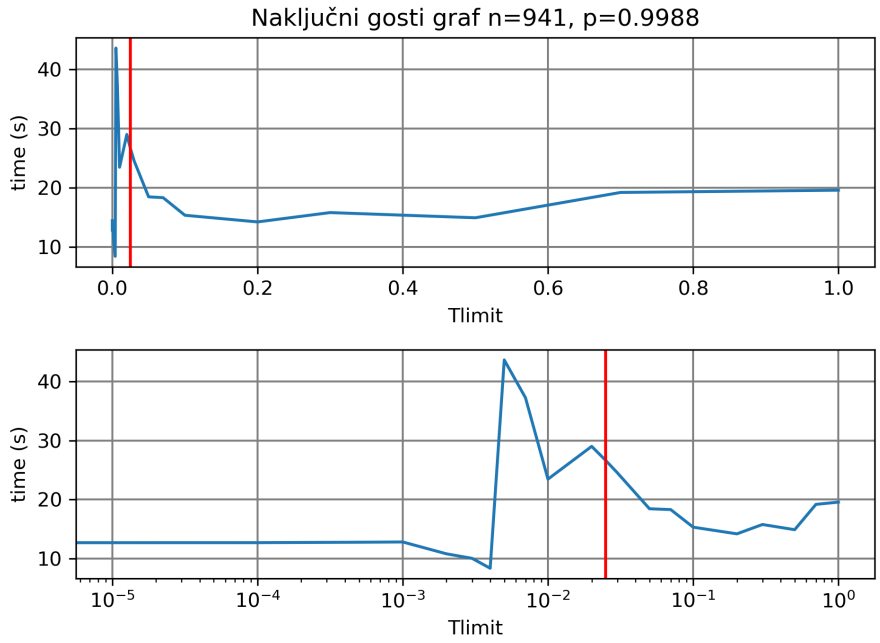
Slika 5.3: Čas v odvisnosti od izbire parametra T_{limit} na grafu iz množice naključnih grafov. Zgoraj na linearni skali parametra T_{limit} in spodaj na logaritemski skali. Rdeča črta predstavlja napoved modela MCQD s privzetim parametrom $T_{limit} = 0,025$.

5.1.5 Proteinski produktni grafi

Na sliki 5.6 so predstavljeni časi, ki jih potrebuje algoritem MCQD, da najde maksimalno kliko na grafu za različne vrednosti T_{limit} parametra. Rdeča črta označuje privzeto vrednost $0,025$. Opazimo, da je privzeta vrednost skoraj optimalna in ne obstaja boljša vrednost parametra T_{limit}, pri kateri bi algoritem MCQD veliko hitreje našel maksimalno kliko.

5.1.6 Proteinski sidrni grafi

Na sliki 5.7 so predstavljeni časi, ki jih potrebuje algoritem MCQD, da najde maksimalno kliko na grafu za različne vrednosti T_{limit} parametra. Rdeča črta označuje privzeto vrednost $0,025$. Opazimo, da privzeta vrednost ni

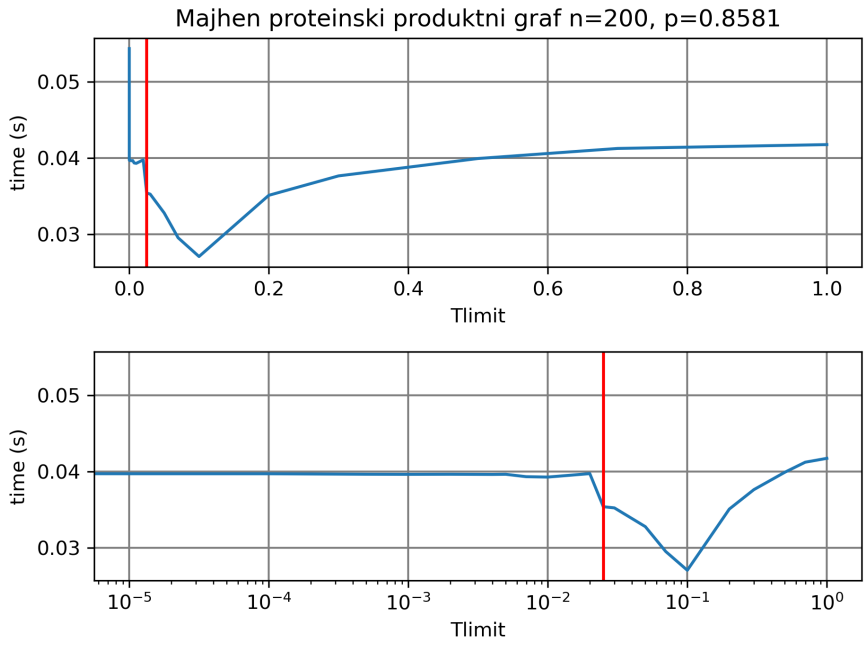


Slika 5.4: Čas v odvisnosti od izbire parametra T_{limit} na grafu iz množice naključnih gostih grafov. Zgoraj na linearni skali parametra T_{limit} in spodaj na logaritemski skali. Rdeča črta predstavlja napoved modela MCQD s privzetim parametrom $T_{limit} = 0,025$.

optimalna in obstaja boljša vrednost parametra T_{limit}, pri kateri bi algoritem MCQD veliko hitreje našel maksimalno kliko.

5.1.7 Grafi DIMACS

Na sliki 5.8 so predstavljeni časi, ki jih potrebuje algoritem MCQD, da najde maksimalno kliko na grafu za različne vrednosti T_{limit} parametra. Rdeča črta označuje privzeto vrednost 0,025. Opazimo, da je privzeta vrednost skoraj optimalna in ne obstaja boljša vrednost parametra T_{limit}, pri kateri bi algoritem MCQD veliko hitreje našel maksimalno kliko.



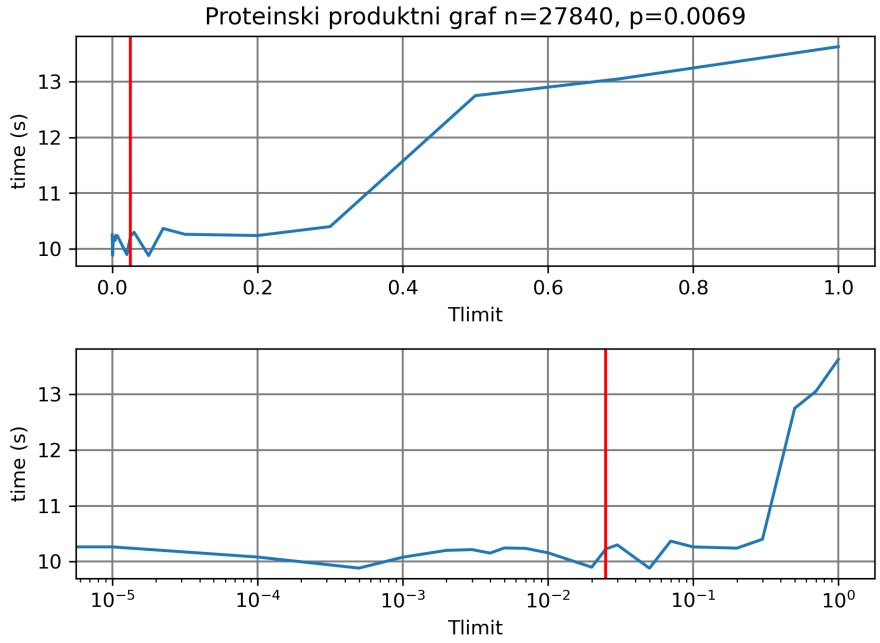
Slika 5.5: Čas v odvisnosti od izbire parametra T_{limit} na grafu iz množice proteinskih grafov. Zgoraj na linearni skali parametra T_{limit} in spodaj na logaritemski skali. Rdeča črta predstavlja napoved modela MCQD s privzetim parametrom $T_{limit} = 0,025$.

5.2 Arhitektura za izvajanje eksperimentov

V tem podpoglavju predstavimo korake in podrobnosti implementacije postopka generiranja grafov, pridobivanja ciljne vrednosti parametra T_{limit} za vsak generiran graf, priprave modelov za strojno učenje ter evalvacije modelov in njihovih napovedi parametra T_{limit}. Na diagramu 5.9 so predstavljeni koraki, ki jih izvedemo za reševanje problema.

5.2.1 Generiranje grafov

Na tem koraku izvedemo generiranje različnih vrst grafov, ki smo jih predstavili v prejšnjem poglavju. Za učenje metod strojnega učenja potrebujemo

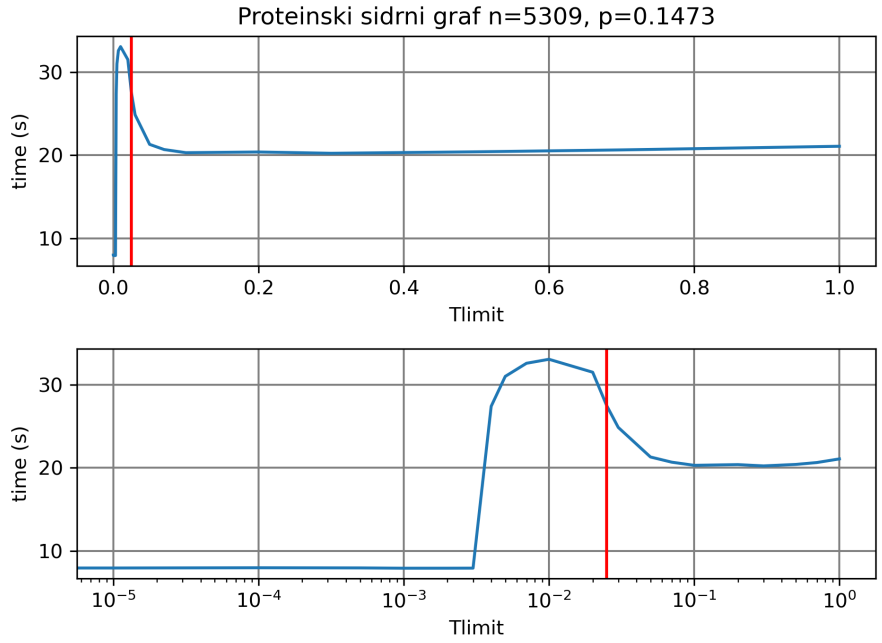


Slika 5.6: Čas v odvisnosti od izbire parametra T_{limit} na grafu iz množice proteinskih produktnih grafov. Zgoraj na linearni skali parametra T_{limit} in spodaj na logaritemski skali. Rdeča črta predstavlja napoved modela MCQD s privzetim parametrom $T_{limit} = 0,025$.

veliko količino raznolikih grafov. Generirali smo 10 000 naključnih grafov do velikosti 500 vozlišč in naključnih gostot $p \in [0, 1]$. Generirali smo 1000 zelo gostih grafov, kjer je $p > 0,99$. Generiramo tudi 300 proteinskih sidrnih grafov in 20 proteinskih produktnih grafov. Proteinski produktni grafi so generirani iz naključnih parov proteinov. Majhni proteinski produktni grafi so generirani iz podmnožice proteinskih produktnih grafov.

5.2.2 Priprava učne in testne množice

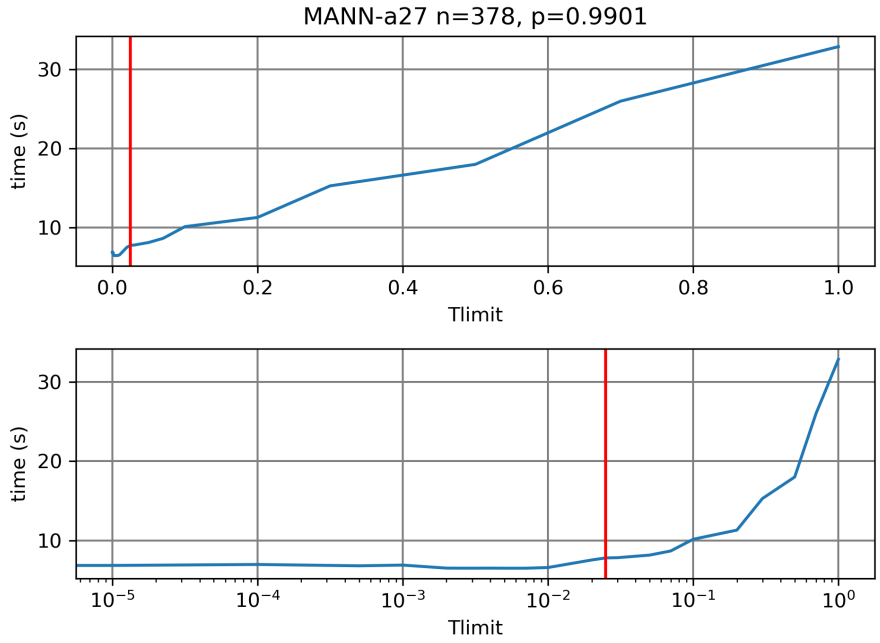
Množico generiranih grafov razdelimo na učne in testne množice za posamezno skupino grafov. Tako dobimo učno množico naključnih, gostih, proteinskih sidrnih in proteinskih produktnih grafov. Testna množica grafov



Slika 5.7: Čas v odvisnosti od izbire parametra T_{limit} na grafu iz množice proteinskih sidrnih grafov. Zgoraj na linearni skali parametra T_{limit} in spodaj na logaritemski skali. Rdeča črta predstavlja napoved modela MCQD s privzetim parametrom $T_{limit} = 0,025$.

je sestavljena iz naključnih grafov, gostih grafov, majhnih proteinskih produktnih grafov, proteinskih sidrnih grafov, proteinskih produktnih grafov in grafov DIMACS.

Vsakemu grafu iz učne množice moramo pripisati vrednost ciljne spremenljivke, ki jo želimo napovedovati. Za vsak graf v učni množici poiščemo približno najboljšo vrednost parametra T_{limit} tako, da za vsak graf poženemo algoritem MCQD za različne vrednosti T_{limit} in shranimo najboljšo. Vrednosti T_{limit} so vzorčene približno enakomerno na logaritemski skali intervala med 0 in 1. Vzorčene vrednosti so 0,0, 0,00001, 0,0001, 0,0005, 0,001, 0,002, 0,003, 0,004, 0,005, 0,007, 0,01, 0,02, 0,025, 0,03, 0,05, 0,07, 0,1, 0,2, 0,3, 0,5, 0,7, 1,0 in so enake za vse grafe. Za vsako vzorčeno vrednost poženemo



Slika 5.8: Čas v odvisnosti od izbire parametra T_{limit} na grafu MANN-a27 iz množice grafov DIMACS. Zgoraj na linearni skali parametra T_{limit} in spodaj na logaritemski skali. Rdeča črta predstavlja napoved modela MCQD s privzetim parametrom $T_{limit} = 0,025$.

algoritem MCQD in zabeležimo, koliko časa potrebuje, da najde maksimalno kliko. Vsakemu grafu iz učne množice pripišemo vrednost parametra T_{limit}, za katero algoritem MCQD najde maksimalno kliko najhitreje.

5.2.3 Učenje modelov strojnega učenja

Pred izvajanjem algoritma MCQD želimo s pomočjo strojnega učenja narediti kar se da dobro napoved parametra T_{limit} tako, da bo izvajanje algoritma MCQD na specifičnem grafu najhitrejše. Za napovedovanje parametra T_{limit} uporabimo več različnih modelov strojnega učenja, ki smo jih predstavili v 3. poglavju. Testiramo privzeti algoritem MCQD, ki ima vrednost parametra T_{limit} enako 0,025 ter izpeljanke tega algoritma, ki namesto privzete vredno-

sti parametra T_{limit} uporabljajo model strojnega učenja, ki napove vrednost parametra T_{limit} pred izvajanjem algoritma MCQD. Uporabimo algoritem XGBoost (XGB), graf konvolucijsko nevronsko mrežo (GCN), graf nevronsko mrežo s pozornostjo (GAT), graf izomorfno nevronsko mrežo (GIN) in metodo podpornih vektorjev z Weisfeiler-Lehmanovo jedrno funkcijo (SVR-WL).

Zaradi velike učne množice in računske zahtevnosti uporabljenih modelov naredimo preprosto validacijo pripravljenih modelov. Modele strojnega učenja učimo na večjem delu (90 %) učne množice grafov. Preostali del (10 %) učne množice uporabimo za validacijo naučenih modelov. Učenje in validacijo vsakega modela izvedemo dvakrat na naključni permutaciji celotne učne množice.

5.2.4 Napovedovanje vrednosti parametra T_{limit}

Po končanem učenju modelov za strojno učenje uporabimo te modele na testni množici grafov. Za vsak graf iz testne množice napovemo vrednost parametra T_{limit} z vsakim naučenim modelom. Tako dobimo za vsak graf iz testnih množic prirejene vrednosti parametra T_{limit} (eno za vsak model).

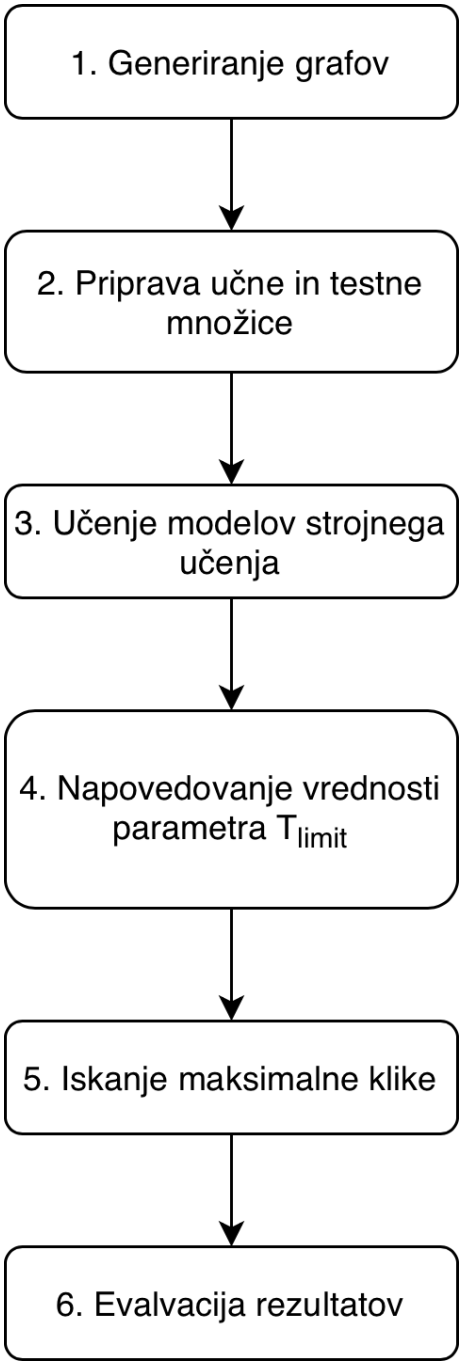
5.2.5 Iskanje maksimalne klike

Na tem koraku izvedemo algoritem MCQD, ki za vsak graf poišče maksimalno kliko z različnimi vrednostmi parametra T_{limit}, ki so jih napovedali naučeni modeli. Po končanem iskanju maksimalne klike si zabeležimo čas iskanja, ki ga primerjamo med različnimi modeli na naslednjem koraku.

5.2.6 Evalvacija rezultatov

Modele za strojno učenje primerjamo za vsak graf tako, da primerjamo čas, ki ga algoritem MCQD porabi za iskanje maksimalne klike z napovedanim parametrom T_{limit}. Evalvacija je izvedena na testnih množicah grafov, ki

vsebujejo raznolike grafe določenega tipa in so uporabljene le pri evalvaciji končnih rezultatov. Rezultati so predstavljeni v naslednjem poglavju.



Slika 5.9: Predstavljeni koraki za izvajanje eksperimentov in evalvacijo rezultatov.

Poglavje 6

Eksperimenti in rezultati

V tem poglavju primerjamo različne modele za strojno učenje v kombinaciji z MCQD algoritmom na različnih množicah grafov. Cilj eksperimentov je ugotoviti, ali je kateri od modelov strojnega učenja zmožen izboljšati čas, v katerem MCQD s privzeto vrednostjo parametra T_{limit} najde maksimalno kliko. Modele primerjamo po času za vsak graf posebej. Časi so predstavljeni v sekundah in izvajanje eksperimentov je omejeno na 2000 sekund. Če algoritem ne najde maksimalne klike v omejenem času, je v tabeli čas enak -1.0000. Krajši čas je boljši. Stolpec n v tabeli predstavlja število vozlišč v grafu in stolpec $p \in [0, 1]$ predstavlja gostoto grafa.

Med seboj primerjamo XGBoost (XGB), graf konvolucijske mreže (GCN), graf nevronske mreže s pozornostjo (GAT), graf izomorfne mreže (GIN) in metodo podpornih vektorjev z Weisfeiler-Lehmanovo jedrno funkcijo (SVR-WL). Algoritem MCQD je uporabljen v prvotni različici z empirično določeno vrednostjo parametra T_{limit} enako 0,025.

Na dnu vsake tabele z rezultati je izračunana pohitritev algoritma MCQD z uporabo napovedanih vrednosti T_{limit} v primerjavi s privzeto vrednostjo 0,025. Pohitritev izračunamo na dva načina. Prvi način upošteva celotni čas, porabljen za iskanje s privzeto vrednostjo parametra na testni množici grafov ter ga deli s celotnim časom, ki ga porabimo za iskanje z napovedano vrednostjo parametra T_{limit}. Drugi način izračuna pohitritve za vsak graf

posebej in nato izračuna povprečje pohitritev za vse grafe v testni množici. Vrednost povprečne pohitritve in skupne pohitritve odraža, kako dobro se izboljšan algoritem primerja s privzetim algoritmom MCQD in vrednostjo parametra $T_{limit} = 0,025$. Vrednost, ki je višja od 1, pomeni, da je izboljšan algoritem hitrejši od privzetega. Višja vrednost pohitritve je boljša. Komplementarne tabele, ki vsebujejo napovedi posameznega modela, se nahajajo v prilogi.

Vsi eksperimenti se izvajajo na računalniku s procesorjem AMD Ryzen 9 3900X 12-Core s frekvenco 2 GHz. Računalnik uporablja 64 GB pomnilnika. Algoritem MCQD uporablja eno samo jedro procesorja.

6.1 Naključni grafi

Kot smo ugotovili v prejšnjem poglavju, so potencialne pohitritve algoritma MCQD na naključnih grafih majhne in relativno redke. V tabeli 6.1 opazimo, da izstopa model GAT, saj dosega hitrejše čase od vseh ostalih modelov, vključno s privzetim MCQD algoritmom. Pohitritve pričakovano niso velike, vendar pa so konsistentne na skoraj vseh grafih v testni množici naključnih grafov. Opazimo, da model XGBoost ne dosega konkurenčnih časov in predvsem pri večjih in gostejših grafih potrebuje od nekaj sekund do nekaj minut več kot ostali modeli, kar je pričakovano, saj atributi, ki jih prejme XGBoost, le grobo opisujejo graf in ne zajamejo njegove topologije tako, kot to naredijo ostali modeli.

Tabela 6.1: Tabela vsebuje čase, ki so jih posamezni algoritmi porabili za iskanje maksimalne klike na vsakem grafu iz testne množice naključnih grafov. Nižja vrednost je boljša.

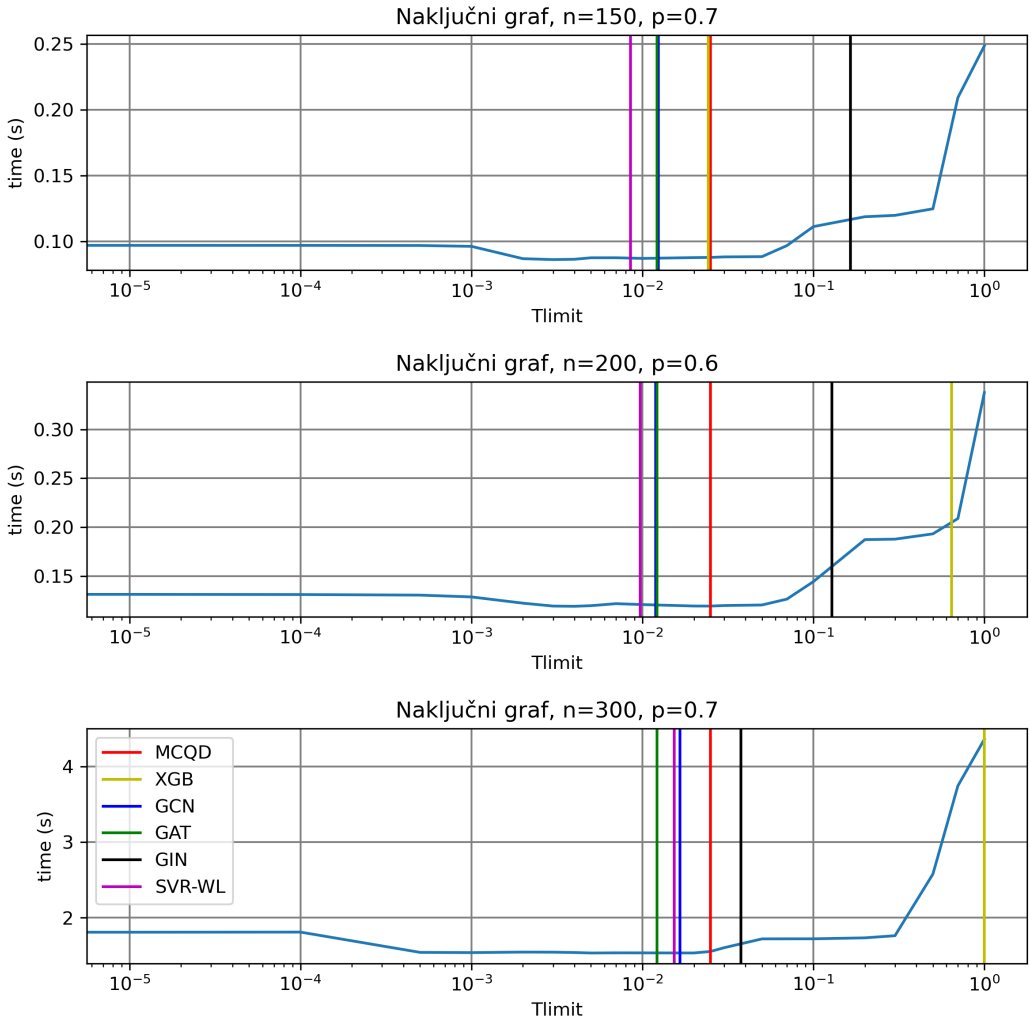
n	**p**	**MCQD**	**XGB**	**GCN**	**GAT**	**GIN**	**SVR-WL**
100	0,6	**0,0029**	**0,0029**	0,0034	**0,0029**	0,0041	**0,0029**
100	0,7	0,0085	0,0084	0,0100	**0,0082**	0,0113	0,0084
100	0,8	0,0152	0,0152	0,0155	**0,0148**	0,0236	0,0196
100	0,9	0,0279	0,0278	0,0287	**0,0274**	0,0378	0,0277

100	0,95	0,0175	0,0168	0,0167	**0,0147**	0,0212	0,0162
150	0,5	0,0057	0,0057	0,0058	**0,0055**	0,0086	**0,0055**
150	0,6	0,0232	0,0230	0,0232	**0,0222**	0,0245	0,0237
150	0,7	0,0858	0,0859	0,0873	**0,0834**	0,1182	0,0837
150	0,8	0,5823	1,6231	0,6503	0,6183	0,6137	**0,5749**
150	0,9	1,3269	3,5726	1,3339	**1,2685**	1,5548	1,3497
150	0,95	**0,6942**	1,0688	0,7400	**0,6942**	0,7737	0,7363
200	0,4	0,0094	0,0090	0,0092	**0,0088**	0,0132	**0,0088**
200	0,5	0,0298	**0,0248**	0,0278	0,0257	0,0294	0,0267
200	0,6	0,1253	0,2039	0,1264	**0,1200**	0,1773	**0,1200**
200	0,7	**0,8442**	1,6124	0,8913	0,8568	0,8841	0,8858
200	0,8	10,9127	30,1734	11,1151	**10,7838**	11,7420	11,3666
200	0,9	146,3930	393,7410	147,4770	**141,0750**	146,1990	143,5000
200	0,95	103,8840	178,0780	108,9580	**102,6420**	103,7040	104,9850
300	0,4	0,0501	0,0509	0,0481	**0,0444**	0,0519	0,0493
300	0,5	0,1880	0,3632	0,1944	**0,1846**	0,1956	0,1856
300	0,6	1,5169	4,1752	1,5431	**1,4630**	1,6790	1,4703
300	0,7	18,5933	48,6900	19,0989	**18,0949**	18,9885	18,2277
500	0,3	0,1038	0,1043	0,0957	**0,0888**	0,1068	0,0977
500	0,4	0,5078	0,9976	0,5280	**0,4983**	0,5248	0,4985
500	0,5	4,8855	6,4727	4,7146	**4,4571**	4,6899	4,4626
500	0,6	51,5837	83,3992	53,1972	**50,4366**	62,5890	57,2126
1000	0,2	0,2645	0,2667	0,2515	0,1938	0,2689	**0,1543**
1000	0,3	1,4628	3,0865	1,4952	1,4331	1,3650	**1,2833**
1000	0,4	26,1727	59,1476	22,7935	**21,3219**	23,5178	21,4335
1000	0,5	386,9080	1248,5100	398,3530	**380,1460**	499,6210	425,9020
skupna pohitritev			0,36	0,97	**1,02**	0,86	0,95
povprečna pohitritev			0,71	0,98	**1,06**	0,88	1,03

Na sliki 6.1 opazimo, da so pohitritve na treh naključnih grafih možne in skoraj vsi modeli dosežejo minimum. Izstopata model XGB, ki z vrednostjo

$T_{limit} = 1$ na vsakem koraku razvrščа vozlišča po njihovi stopnji, in model GIN, ki ne napove optimalne vrednosti parametra.

Iz zadnjih dveh vrstic v tabeli 6.1 opazimo, da najvišjo skupno in povprečno pohitritev doseže model GAT. V povprečju je hitrejši za 6 %. Ostali modeli so počasnejši kot privzeti algoritem MCQD.



Slika 6.1: Prikaz napovedi parametra T_{limit} za različne modele na naključnih grafih.

6.2 Gosti naključni grafi

Na testni množici gostih naključnih grafov, ki je nekakšna razširitev prejšnje skupine naključnih grafov, opazimo, da povprečno najboljše rezultate dosega model GAT, vendar pa so možnosti za pohitritve očitne predvsem na primerih, kjer ostali algoritmi dosegajo boljše pohitritve kot model GAT. Na primeru grafa s 690 vozlišči in gostoto 0,9978 opazimo, da je čas modela GIN več kot 200 s manjši od MCQD in več kot 300 s manjši od modela GAT. Opazimo, da model GIN ne najde maksimalne klike v manj kot 2000 sekundah na grafu z 828 vozlišči in gostoto 0,9979. Vsem ostalim modelom uspe najti maksimalno kliko v omejenem času.

Na sliki 6.2 opazimo, da GCN in GAT dajeta podobne napovedi na vseh grafih. Opazimo tudi, da je nižja vrednost parametra T_{limit} boljša na tej množici grafov.

6.3 Majhni proteinski produktni grafi

Na testni množici majhnih proteinskih produktnih grafov opazimo, da privzeti algoritem MCQD dosega dobre rezultate, primerljive z ostalimi modeli, vendar ni nikoli najhitrejši. Največ pohitritev doseže model GAT. Ostali modeli ne dosegajo konkurenčnih časov na vseh grafih. Primer zahtevnega grafa je graf z 905 vozlišči, kjer sta modela GIN in SVR-WL izredno počasna v primerjavi z ostalimi modeli. Opazimo, da je časovna razlika med modeloma XGB in GIN na grafu z 905 vozlišči zelo velika v primerjavi z ostalimi modeli.

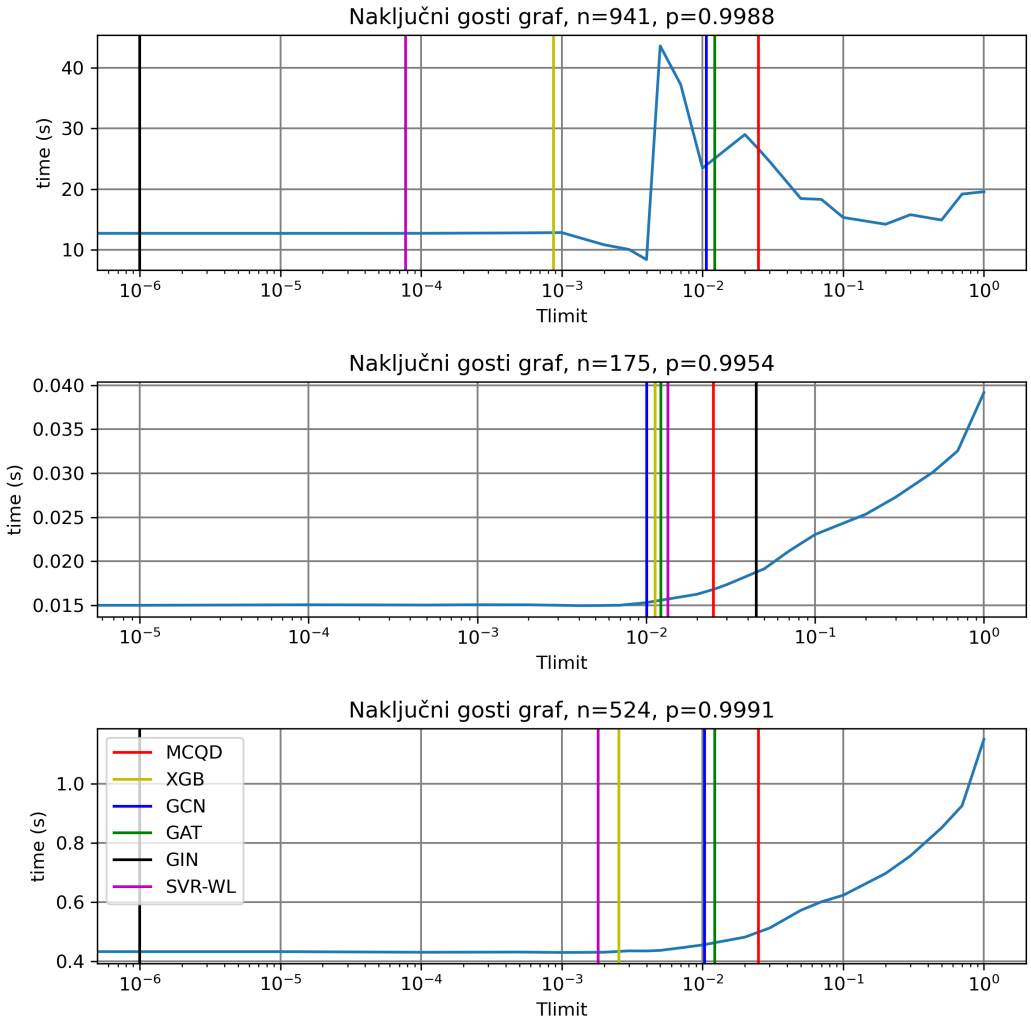
Na sliki 6.3 opazimo, da je optimalna izbira parametra T_{limit} lahko zelo različna za grafe iz množice majhnih proteinskih produktnih grafov. Nekateri grafi zahtevajo razvrščanje vozlišč na vsakem koraku (na primer graf z $n = 563$ in $p = 0,9800$), drugi ne potrebujejo razvrščanja za hitro iskanje maksimalne klike (na primer graf z $n = 346$ in $p = 0,9091$).

Tabela 6.2: Tabela vsebuje čase, ki so jih posamezni algoritmi porabili za iskanje maksimalne klike na vsakem grafu iz testne množice naključnih gostih grafov. Nižja vrednost je boljša.

n	**p**	**MCQD**	**XGB**	**GCN**	**GAT**	**GIN**	**SVR-WL**
63	0,9944	0,0008	**0,0007**	**0,0007**	**0,0007**	0,0011	**0,0007**
113	0,9987	0,0024	**0,0022**	0,0023	**0,0022**	0,0028	0,0023
121	0,9955	0,0044	0,0042	0,0042	**0,0041**	0,0065	0,0047
175	0,9954	0,0171	0,0159	0,0157	**0,0151**	0,0194	0,0157
304	0,9911	8,8271	6,4638	7,3050	**6,2747**	8,6368	9,3515
414	0,9943	2,3677	1,8574	1,7514	**1,2559**	5,0611	1,9631
443	0,9938	57,8980	**55,2395**	66,4033	58,8421	428,4730	265,8170
475	0,9979	0,2406	0,2327	0,2413	0,2305	0,2336	**0,2287**
476	0,9977	0,3262	0,2695	0,3024	0,2906	0,2703	**0,2652**
524	0,9992	0,5042	0,4380	0,4660	0,4482	0,4341	**0,4278**
622	0,9981	0,6802	0,6225	0,6212	**0,6082**	0,6253	0,6120
690	0,9978	326,0520	1124,6500	511,1010	428,9200	**115,9220**	-1,0000
828	0,9979	382,5500	322,8460	431,3020	**254,8100**	-1,0000	1217,8400
931	0,9995	1,9800	1,7438	1,7799	1,7807	1,7584	**1,7017**
941	0,9988	25,4684	12,2125	22,7202	20,2954	12,3739	**12,0440**
skupna pohitritev			0,52	0,77	**1,04**	0,73	0,31
povprečna pohitritev			1,14	1,04	**1,18**	1,09	1,05

6.4 Proteinski produktni grafi

Ker so proteinski produktni grafi izredno veliki in redki, smo uporabili le tri generirane grafe v testni množici. Opazimo, da vsi modeli dosegajo podobne čase in so razlike med njimi minimalne. Iz tega naključnega vzorca proteinskih produktnih grafov lahko sklepamo, da so velike pohitritve algoritma MCQD na takšnih redkih grafih nedosegljive.



Slika 6.2: Prikaz napovedi parametra T_{limit} za različne modele na gostih grafih.

Tabela 6.4: Tabela vsebuje čase, ki so jih posamezni algoritmi porabili za iskanje maksimalne klike na vsakem grafu iz testne množice proteinskih produktnih grafov. Nižja vrednost je boljša.

n	**p**	**MCQD**	**XGB**	**GCN**	**GAT**	**GIN**	**SVR-WL**
27840	0,0069	9,8018	**9,9147**	10,2909	9,8759	10,97434	10,1547
36841	0,0060	**18,6482**	19,7002	19,1695	19,09	23,4188	19,9433
121359	0,0024	**198,592**	199,500	199,417	199,852	378,121	199,348
skupna pospešitev			0,99	0,99	0,99	0,55	0,99
povprečna pospešitev			0,98	0,98	0,99	0,74	0,97

Tabela 6.3: Tabela vsebuje čase, ki so jih posamezni algoritmi porabili za iskanje maksimalne klike na vsakem grafu iz testne množice majhnih proteinskih produktnih grafov. Nižja vrednost je boljša.

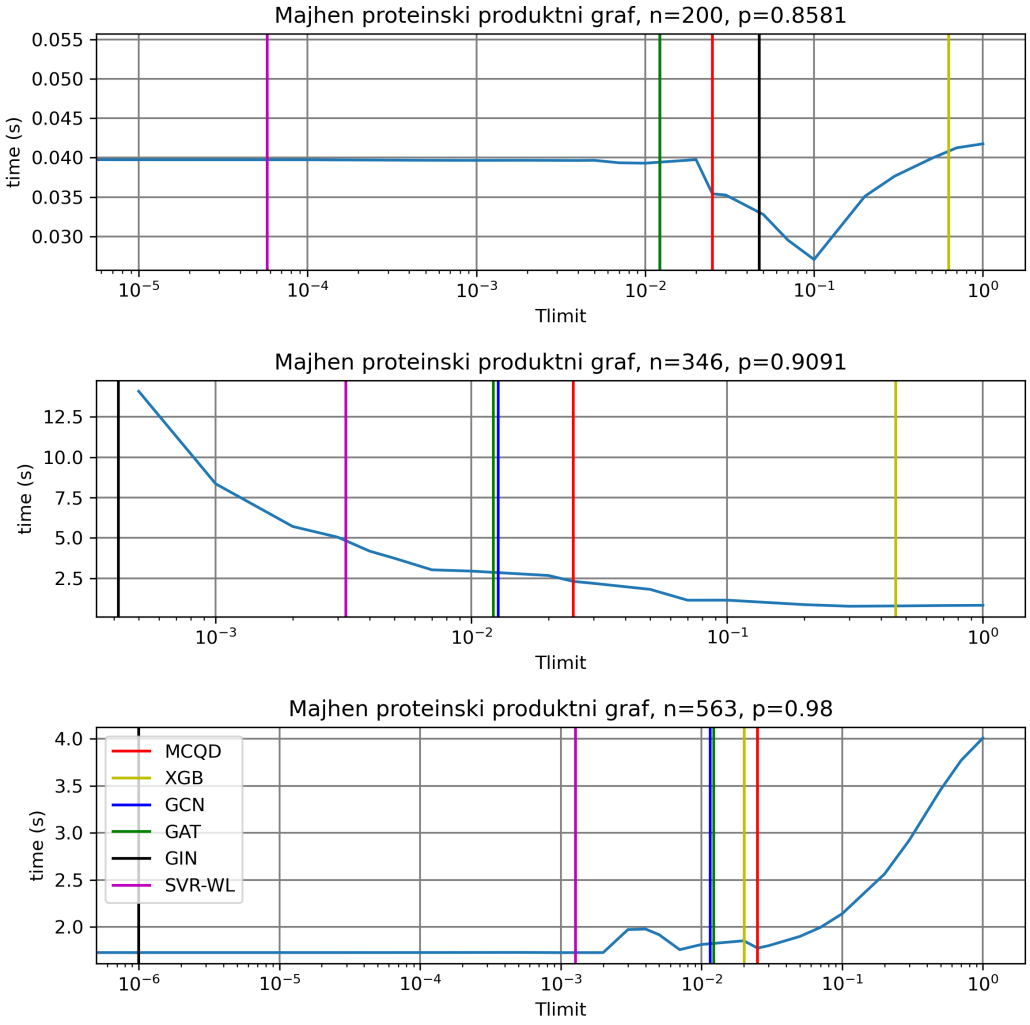
n	**p**	**MCQD**	**XGB**	**GCN**	**GAT**	**GIN**	**SVR-WL**
61	0,9792	0,0008	0,0008	0,0008	**0,0007**	0,0012	0,0008
138	0,9422	0,0079	0,0137	0,0078	**0,0074**	0,0102	0,0076
200	0,8581	0,0358	0,0398	0,0388	0,0381	**0,0327**	0,0393
271	0,9852	0,2062	0,2004	0,1972	0,1907	**0,1831**	0,1913
346	0,9091	2,3032	**0,7774**	2,8878	2,8278	14,3173	4,7920
451	0,9743	0,8956	0,8989	0,8955	**0,8464**	1,3257	1,3406
563	0,9800	1,7685	1,8496	1,7348	**1,6936**	1,7277	1,6994
655	0,9692	2,3652	2,3684	**2,4533**	2,6894	15,9674	15,8806
750	0,9625	4,7147	5,8504	4,2834	**4,1741**	8,0964	8,0182
905	0,9412	18,4683	**16,2290**	25,2455	18,5778	-1,0000	283,5820
skupna pohitritev			**1,08**	0,81	0,99	0,29	0,09
povprečna pohitritev			**1,13**	0,96	1,02	0,69	0,70

Na sliki 6.4 opazimo, da različni modeli dosežejo podobne čase za iskanje maksimalne klike na naključno izbranih proteinskih produktnih grafih. Odstopa le model GIN, ki ne med učenjem modela, ki zaradi majhne učne množice ne ustvari ustrezne reprezentacije proteinskega produktnega grafa in zato ne najde ustrezne vrednosti parametra T_{limit}.

6.5 Proteinski sidrni grafi

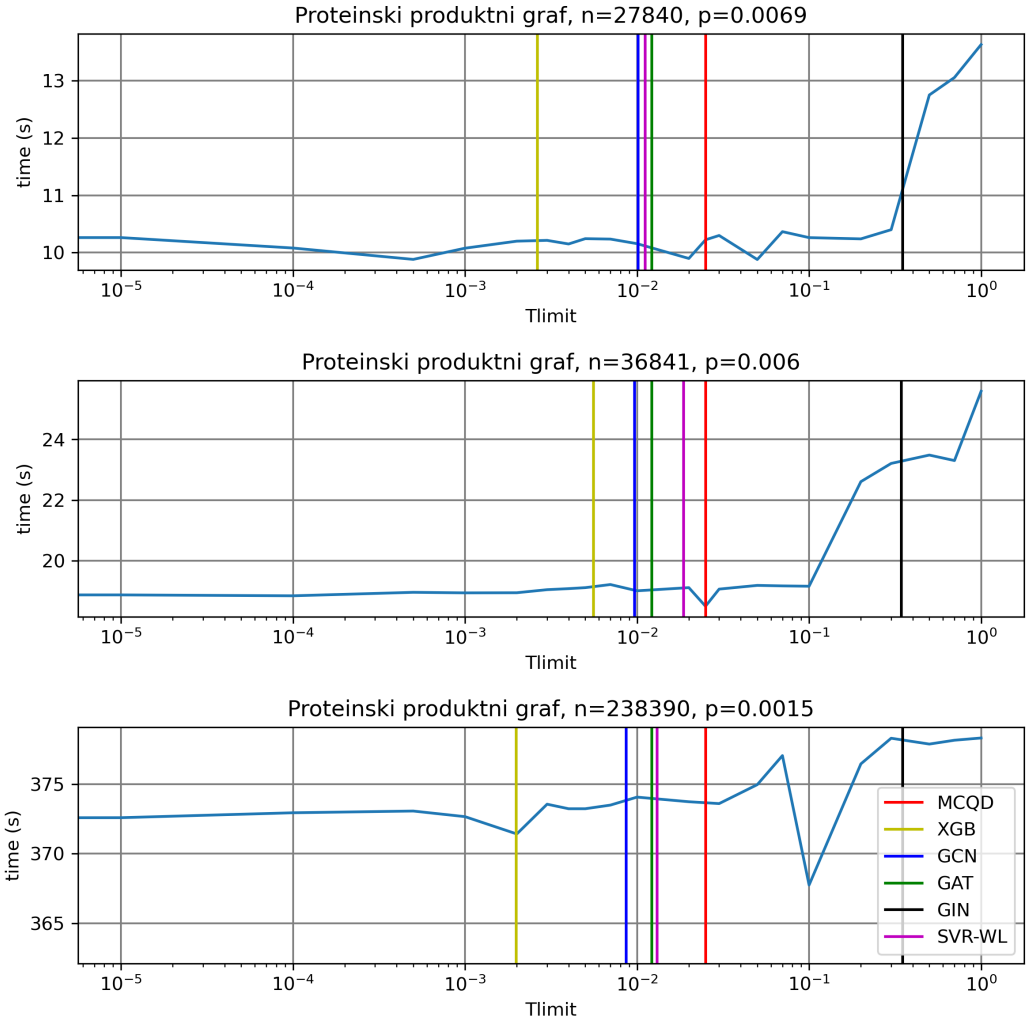
Na testni množici proteinskih sidrnih grafov opazimo, da so proteinski sidrni grafi relativno redki in je pričakovani čas iskanja maksimalne klike sorazmerno majhen. Na večini grafov je ponovno najhitrejši model GAT, s katerim je povprečna pohitritev približno 34 %. Izkaže se tudi model SVR-WL, ki dosega podobne pohitritve.

Na sliki 6.5 opazimo, da je na naključno izbranih grafih iz množice pro-



Slika 6.3: Prikaz napovedi parametra T_{limit} za različne modele na proteinskem grafu.

teinskih sidrnih grafov najboljša izbira parametra T_{limit} enaka 0. Na grafu $n = 5309$ je možna več kot trikratna pohitritev iskanja maksimalne klike, kot je razvidno iz tabele 6.5 in slike 6.5.



Slika 6.4: Prikaz napovedi parametra T_{limit} za različne modele na proteinskih produktnih grafih.

6.6 Grafi DIMACS

Grafi DIMACS so sintetična množica grafov, ki se pogosto uporablja za primerjanje algoritmov za iskanje maksimalne klike. V tabeli 6.6 opazimo, da je prvotni algoritem MCQD z vrednostjo parametra T_{limit} skoraj vedno najhitrejši. Tak rezultat je pričakovan glede na sestavo množice grafov DIMACS.

Tabela 6.5: Tabela vsebuje čase, ki so jih posamezni algoritmi porabili za iskanje maksimalne klike na vsakem grafu iz testne množice proteinskih sidrnih grafov. Nižja vrednost je boljša.

n	p	MCQD	XGB	GCN	GAT	GIN	SVR-WL
345	0,1266	**0,0025**	**0,0025**	0,0026	**0,0025**	0,0026	**0,0025**
1779	0,1108	0,0940	0,0948	0,0943	**0,0939**	0,09517	0,0952
1851	0,1580	0,1606	0,1606	0,1829	**0,1562**	0,4394	0,1580
3233	0,1620	3,6941	3,4489	1,9791	1,9817	3,5981	**1,9176**
4211	0,0448	0,3889	0,3900	0,3990	**0,3783**	0,3967	0,3823
5293	0,1119	2,7147	6,0876	2,9925	**2,6810**	5,4606	2,7374
5309	0,1474	26,3695	19,1478	33,7628	7,7648	7,8752	**7,5596**
5735	0,0592	1,2673	1,2681	1,3803	**1,2271**	1,3196	1,2476
6294	0,1382	3,0941	15,8609	3,3343	**3,0363**	3,1517	3,0399
7211	0,1012	4,4230	11,2341	4,5631	**4,2580**	9,0498	4,3415
skupna pospešitev			0,73	0,86	**1,96**	1,41	**1,96**
povprečna pospešitev			0,84	1,01	**1,34**	1,10	**1,34**

Ker so grafi DIMACS zelo raznoliki in jih ne uporabljamo v učni množici grafov, so rezultati za algoritme strojnega učenja pričakovano slabši.

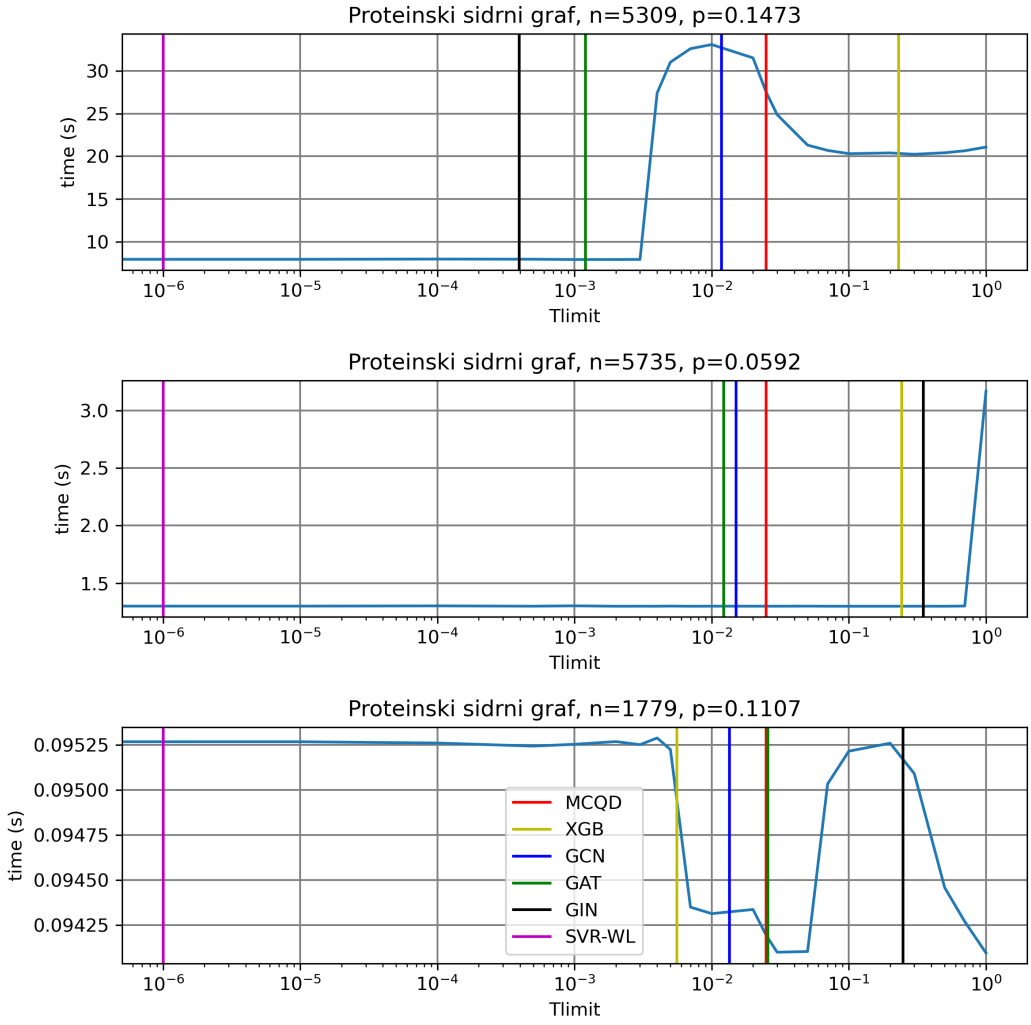
Tabela 6.6: Tabela vsebuje čase, ki so jih posamezni algoritmi porabili za iskanje maksimalne klike na vsakem grafu iz testne množice grafov DIMACS. Nižja vrednost je boljša.

Ime grafa	MCQD	XGB	GCN	GAT	GIN	SVR-WL
MANN-a27	8,1881	7,9004	**6,8391**	7,0559	7,2123	7,2626
MANN-a45	-1,0000	-1,0000	-1,0000	-1,0000	-1,0000	-1,0000
MANN-a81	-1,0000	-1,0000	-1,0000	-1,0000	-1,0000	-1,0000
MANN-a9	**0,0003**	0,0008	**0,0003**	**0,0003**	0,0007	**0,0003**
brock200-1	**1,5104**	1,5271	1,5796	1,5206	1,5701	1,7254
brock200-2	0,0202	0,0354	0,0188	**0,0180**	0,0234	0,0190
brock200-3	**0,0724**	0,1479	0,0756	0,0736	0,0761	0,0751

brock200-4	**0,2258**	0,6799	0,2350	0,2285	0,3164	0,2969
brock400-1	**979,26**	1224,94	1021,78	993,49	1257,55	1049,75
brock400-2	466,72	516,48	484,22	**465,02**	530,32	491,07
brock400-3	**845,14**	-1,0000	862,34	857,08	1101,45	866,54
brock400-4	**469,47**	1437,98	475,58	476,26	571,65	563,04
brock800-1	-1,0000	-1,0000	-1,0000	-1,0000	-1,0000	-1,0000
brock800-2	-1,0000	-1,0000	-1,0000	-1,0000	-1,0000	-1,0000
brock800-3	-1,0000	-1,0000	-1,0000	-1,0000	-1,0000	-1,0000
brock800-4	-1,0000	-1,0000	-1,0000	-1,0000	-1,0000	-1,0000
c-fat200-1	0,0007	0,0007	0,0007	0,0007	0,0007	0,0009
c-fat200-2	**0,0009**	0,0018	0,0010	0,0010	0,0011	0,0010
c-fat200-5	**0,0028**	0,0069	0,0030	0,0029	0,0036	**0,0028**
c-fat500-1	**0,0033**	**0,0033**	0,0034	0,0034	0,0034	0,0040
c-fat500-10	0,0266	0,0262	0,0275	0,0270	0,0278	**0,0260**
c-fat500-2	**0,0040**	0,0041	0,0042	0,0042	0,0044	0,0066
c-fat500-5	**0,0090**	0,0213	0,0093	0,0093	0,0103	0,0094
hamming10-2	4,8228	88,2296	2,6917	2,9932	1,1188	**1,0914**
hamming10-4	-1,0000	-1,0000	-1,0000	-1,0000	-1,0000	-1,0000
hamming6-2	0,0004	0,0004	0,0004	0,0004	0,0004	0,0004
hamming6-4	0,0002	0,0002	0,0002	0,0002	0,0002	0,0002
hamming8-2	0,0206	0,0547	0,0183	0,0186	0,0175	**0,0169**
hamming8-4	**0,1119**	0,4121	0,1132	0,1135	0,1992	0,2758
johnson16-2-4	0,8713	1,2958	**0,7449**	0,7549	1,2916	0,8990
johnson32-2-4	-1,0000	-1,0000	-1,0000	-1,0000	-1,0000	-1,0000
johnson8-2-4	0,0001	0,0001	0,0001	0,0001	0,0001	0,0001
johnson8-4-4	**0,0008**	0,0029	**0,0008**	**0,0008**	0,0022	**0,0008**
keller4	**0,0383**	0,0384	0,0394	0,0399	0,0678	0,0902
keller5	-1,0000	-1,0000	-1,0000	-1,0000	-1,0000	-1,0000
keller6	-1,0000	-1,0000	-1,0000	-1,0000	-1,0000	-1,0000
p-hat1000-1	0,8662	0,9141	0,8884	0,9263	0,7837	**0,7742**
p-hat1000-2	**650,00**	902,27	655,43	687,32	1592,85	1075,80

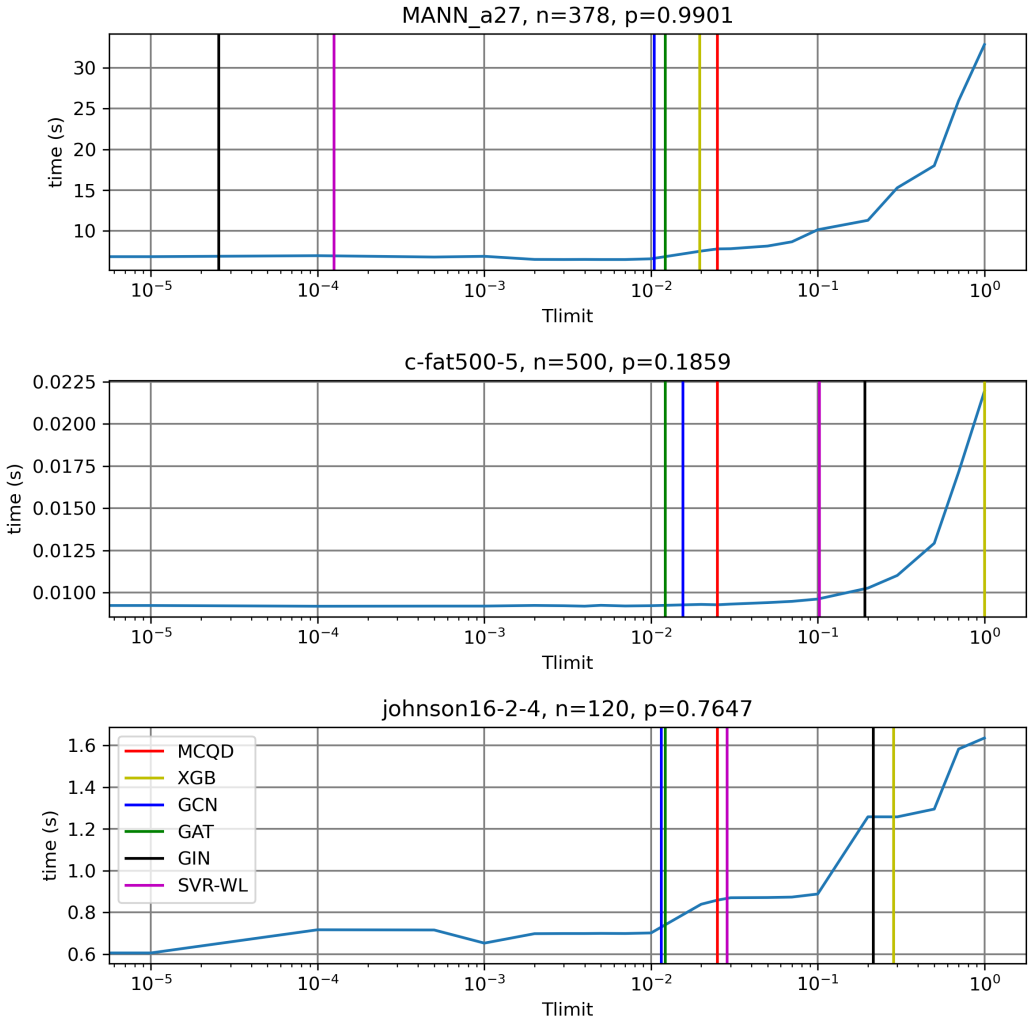
p-hat1000-3	-1,0000	-1,0000	-1,0000	-1,0000	-1,0000	-1,0000
p-hat1500-1	6,3084	6,6570	6,6594	6,7186	**6,2424**	6,2625
p-hat1500-2	-1,0000	-1,0000	-1,0000	-1,0000	-1,0000	-1,0000
p-hat1500-3	-1,0000	-1,0000	-1,0000	-1,0000	-1,0000	-1,0000
p-hat300-1	**0,0063**	0,0118	0,0066	0,0067	0,0112	0,0066
p-hat300-2	**0,0545**	0,1561	0,0564	0,0572	0,0635	0,0565
p-hat300-3	**7,9488**	27,4458	8,2173	8,6583	8,7833	10,6970
p-hat500-1	**0,0407**	0,0430	0,0428	0,0434	0,0653	0,0428
p-hat500-2	**2,3609**	2,4558	2,4427	2,4774	2,4437	2,4428
p-hat500-3	**564,7910**	590,8150	585,0510	597,5300	1045,7800	590,1180
p-hat700-1	0,1867	0,1476	0,1929	0,1490	0,2017	**0,1473**
p-hat700-2	**19,7820**	33,3202	20,4586	20,7683	29,2213	22,7913
p-hat700-3	-1,0000	-1,0000	-1,0000	-1,0000	-1,0000	-1,0000
san1000	**1,2739**	1,3645	1,3469	1,3506	16,0314	16,3943
san200-0,7-1	0,0131	0,0244	0,0126	0,0126	0,0185	**0,0121**
san200-0,7-2	0,0184	0,0461	**0,0183**	0,0194	0,0248	0,0185
san200-0,9-1	0,1926	0,2760	0,2177	0,2170	**0,1867**	0,2299
san200-0,9-2	**1,0508**	3,1562	1,0740	1,0696	1,0721	1,0902
san200-0,9-3	**5,9893**	16,3786	6,1211	6,1670	6,3727	6,0911
san400-0,5-1	0,0274	0,0618	0,0299	**0,0267**	0,0270	0,0619
san400-0,7-1	**0,7009**	1,2987	0,7657	0,7997	0,7785	0,7965
san400-0,7-2	**0,2582**	0,3043	0,3327	0,3798	1,4452	0,9645
san400-0,7-3	4,0471	9,1135	4,0360	**3,9668**	5,2917	4,8431
san400-0,9-1	48,261	114,540	48,781	49,023	68,488	**47,119**
sanr200-0,7	**0,5775**	1,7081	0,6125	0,6153	0,6316	0,6234
sanr200-0,9	**89,044**	272,944	93,206	93,572	89,477	123,953
sanr400-0,5	**1,0750**	3,3913	1,1330	1,1365	1,0934	1,1198
sanr400-0,7	**269,480**	300,851	305,288	304,624	295,005	302,635
skupna pospešitev		0,64	0,96	0,96	0,66	0,85
povprečna pospešitev		0,63	0,99	0,98	0,85	0,94

Na sliki 6.6 so predstavljene napovedi različnih modelov za grafe MANN-



Slika 6.5: Prikaz napovedi parametra T_{limit} za različne modele na proteinskih sidrnih grafih.

a27, c-fat500 in johnson16-2-4 iz množice grafov DIMACS. Opazimo, da se na teh treh grafih urejanje vozlišč ne izplača in je najbolj primerna vrednost parametra T_{limit} enaka 0.



Slika 6.6: Prikaz napovedi parametra T_{limit} za različne modele na grafih DIMACS.

6.7 Uteženi proteinski sidrni grafi

Model GAT, ki se je najbolje odrezal na proteinskih sidrnih grafih, v tabeli 6.7 primerjamo še na uteženih proteinskih sidrnih grafih. Ti grafi so enaki grafom iz testne množice proteinskih sidrnih grafov, z razliko dodanih uteži na vsako vozlišče grafa. Opazimo, da iskanje utežene klike potrebuje nekoliko več časa

kot iskanje neuteženе klike na istem grafu. Na večini grafov je hitrejši model GAT, vendar pa so pohitritve v primerjavi s privzetim modelom MCQD relativno majhne.

Tabela 6.7: Tabela vsebuje čase, ki so jih posamezni algoritmi porabili za iskanje uteženе maksimalne klike na vsakem grafu iz testne množice uteženih proteinskih sidrnih grafov. Nižja vrednost je boljša.

n	**p**	**MCQD**	**GAT**
345	0,1266	0,0049	0,00483
1779	0,1108	0,1188	0,1122
1851	0,1580	1,0636	1,0563
3233	0,1620	107,3550	106,3990
4211	0,0448	0,4950	0,4960
5293	0,1119	1,1551	1,1550
5309	0,1474	16,5672	16,4841
5735	0,0592	1,3452	1,3416
6294	0,1382	11,4437	10,8346
7211	0,1012	6,8934	6,88820
skupna pohitritev			1,01
povprečna pohitritev			1,01

6.8 Interpretacija rezultatov

Iz rezultatov različnih eksperimentov lahko ugotovimo, da lahko na veliki večini grafov uporabimo privzeti algoritem MCQD, saj dosega konkurenčne rezultate na vseh predstavljenih množicah grafov. Največje potencialne izboljšave smo opazili na naključnih gostih grafih in proteinskih sidrnih grafih z modelom GAT.

Ugotovimo tudi, da imajo lahko različni grafi iz iste podatkovne množice (podobna struktura) zelo različne vrednosti optimalnega parametra T_{limit} in

da začetno razvrščanje vozlišč v povprečju bistveno ne vpliva na čas, potreben za iskanje maksimalne klike.

Iz rezultatov je razvidno, da nekateri modeli delajo zelo podobne napovedi. Primer sta GCN in GAT, ki dajeta zelo podobne napovedi na različnih vrstah grafov. Iz priloženih tabel A.2 in A.4 opazimo, da model GAT napoveduje enako vrednost za vse grafe. Iz tega lahko sklepamo, da se je model GAT naučil napovedovati enako vrednost za družine grafov, ki jih je v učni množici zelo malo oziroma jih sploh ni.

Ugotovimo, da je privzeti algoritem MCQD hiter na vseh predstavljenih skupinah grafov, vendar so izboljšave možne z dobro izbiro parametra T_{limit}. Predvsem na naključnih, gostih naključnih in proteinskih sidrnih grafih so pohitritve dosežene z uporabo modela GAT. Ugotovimo, da model XGB ne dosega konkurenčnih rezultatov zaradi nizke vsebovane informacije v izbranih značilkah. Prav tako model SVR-WL ne dosega dobrih rezultatov in je pogosto nekonsistenten v kakovosti svojih napovedi parametra T_{limit}. Izstopa tudi model GIN, ki se zaradi težavnosti učenja na manjših učnih množicah ne nauči dovolj dobre reprezentacije grafa za napovedovanje parametra T_{limit}.

Iz rezultatov proteinskih produktnih grafov in DIMACS grafov lahko sklepamo, da je velikost in kakovost učne množice iz iste domene grafov ključna za doseganje dobrih rezultatov z modeli strojnega učenja.

Iz eksperimenta z uteženimi sidrnimi grafi opazimo, da je na praktičnih primerih smiselno uporabiti privzeti algoritem MCQD, saj so pohitritve z modelom GAT zanemarljive (približno 1 % pohitritve).

Poglavje 7

Zaključki

V magistrskem delu smo predstavili problem maksimalne klike in algoritme za reševanje tega problema. Podrobneje smo opisali algoritem MCQD, ki ga nadgradimo z dodatnim korakom strojnega učenja na grafu. Predstavili smo uporabljene algoritme strojnega učenja in podatkovne množice za učenje in testiranje izboljšanega algoritma. Preverili smo tudi možnosti za pohitritev prvotnega algoritma MCQD na vsaki podatkovni množici. Podrobno smo predstavili postavitev eksperimentov in korake za učinkovito izvajanje učenja in evalvacije algoritmov strojnega učenja.

Ugotovili smo, da so pohitritve prvotnega algoritma MCQD možne na naključnih, gostih naključnih, proteinskih sidrnih in uteženih proteinskih sidrnih grafih. Na množicah majhnih proteinskih produktnih grafov, proteinskih produktnih grafov in grafov DIMACS so pohitritve minimalne oziroma jih sploh ni.

Za praktično uporabo bi uporabili model GAT na naključnih, gostih naključnih in uteženih proteinskih sidrnih grafih. Za ostale množice grafov pa bi uporabili privzeti algoritem MCQD, ki z zelo robustno izbiro parametra T_{limit} hitro najde maksimalno kliko. Z uporabo modela GAT bi pohitrili iskanje maksimalne klike na račun dodatne kompleksnosti algoritma MCQD, kateremu moramo dodati še korak napovedovanja parametra.

Pri nadaljnjem delu bi bilo primerno povečati podatkovno množico pro-

teinov in združiti napovedi najboljših modelov tako, da bi pridobili razširjen algoritem MCQD, ki bi bil primeren za praktično uporabo in bi deloval bolj zanesljivo na množici proteinskih grafov. Zanimivo bi bilo preveriti, ali lahko dosežemo pohitritve algoritma MCQD na kakšni drugi domeni grafov. Pristope, ki smo jih uporabili za razširitev in izboljšavo algoritma MCQD, lahko uporabimo tudi za razširitev drugih algoritmov, ki delujejo na grafih in uporabljajo empirično določene vrednosti parametrov algoritma.

Literatura

[1] Q. Wu, J.-K. Hao, A review on algorithms for maximum clique problems, European Journal of Operational Research 242 (3) (2015) 693–709 (2015).

[2] M. Depolli, J. Konc, K. Rozman, R. Trobec, D. Janezic, Exact parallel maximum clique algorithm for general and protein graphs, Journal of chemical information and modeling 53 (9) (2013) 2217–2228 (2013).

[3] S. Butenko, W. E. Wilhelm, Clique-detection models in computational biochemistry and genomics, European Journal of Operational Research 173 (1) (2006) 1–17 (2006).

[4] J. Konc, D. Janezic, An improved branch and bound algorithm for the maximum clique problem, proteins 4 (5) (2007).

[5] M. Prates, P. H. Avelar, H. Lemos, L. C. Lamb, M. Y. Vardi, Learning to solve NP-complete problems: A graph neural network for decision TSP, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 33, 2019, pp. 4731–4738 (2019).

[6] J. L. Walteros, A. Buchanan, Why is maximum clique often easy in practice?, Operations Research (2020).

[7] C.-M. Li, H. Jiang, F. Manyà, On minimization of the number of branches in branch-and-bound algorithms for the maximum clique problem, Computers & Operations Research 84 (2017) 1–15 (2017).

[8] E. Tomita, T. Seki, An efficient branch-and-bound algorithm for finding a maximum clique, in: International conference on discrete mathematics and theoretical computer science, Springer, 2003, pp. 278–289 (2003).

[9] E. Tomita, S. Matsuzaki, A. Nagao, H. Ito, M. Wakatsuki, A much faster algorithm for finding a maximum clique with computational experiments, Journal of Information Processing 25 (2017) 667–677 (2017).

[10] R. Carraghan, P. M. Pardalos, An exact algorithm for the maximum clique problem, Operations Research Letters 9 (6) (1990) 375–382 (1990).

[11] C.-M. Li, Z. Quan, An efficient branch-and-bound algorithm based on maxsat for the maximum clique problem, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 24, 2010 (2010).

[12] P. San Segundo, A. Lopez, P. M. Pardalos, A new exact maximum clique algorithm for large and massive sparse graphs, Computers & Operations Research 66 (2016) 81–94 (2016).

[13] Y. Bengio, A. Lodi, A. Prouvost, Machine learning for combinatorial optimization: a methodological tour d'horizon, European Journal of Operational Research (2020).

[14] K. Abe, Z. Xu, I. Sato, M. Sugiyama, Solving NP-hard problems on graphs with extended AlphaGo Zero, arXiv preprint arXiv:1905.11623 (2019).

[15] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, M. Sun, Graph neural networks: A review of methods and applications, arXiv preprint arXiv:1812.08434 (2018).

[16] C. Cortes, V. Vapnik, Support-vector networks, Machine learning 20 (3) (1995) 273–297 (1995).

[17] J. H. Friedman, Greedy function approximation: a gradient boosting machine, Annals of statistics (2001) 1189–1232 (2001).

[18] T. Chen, C. Guestrin, Xgboost: A scalable tree boosting system, in: Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining, 2016, pp. 785–794 (2016).

[19] N. Shervashidze, S. Vishwanathan, T. Petri, K. Mehlhorn, K. Borgwardt, Efficient graphlet kernels for large graph comparison, in: Artificial intelligence and statistics, PMLR, 2009, pp. 488–495 (2009).

[20] N. Shervashidze, P. Schweitzer, E. J. Van Leeuwen, K. Mehlhorn, K. M. Borgwardt, Weisfeiler-Lehman graph kernels., Journal of Machine Learning Research 12 (9) (2011).

[21] J. Leskovec. Stanford cs224w: Machine learning with graphs. Traditional methods for machine learning in graphs [online] (2019). Accessed 2021-5-1.

[22] K. Xu, W. Hu, J. Leskovec, S. Jegelka, How powerful are graph neural networks?, arXiv preprint arXiv:1810.00826 (2018).

[23] E. Khalil, H. Dai, Y. Zhang, B. Dilkina, L. Song, Learning combinatorial optimization algorithms over graphs, in: Advances in Neural Information Processing Systems, 2017, pp. 6348–6358 (2017).

[24] T. N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, arXiv preprint arXiv:1609.02907 (2016).

[25] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, I. Polosukhin, Attention is all you need, arXiv preprint arXiv:1706.03762 (2017).

[26] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, Y. Bengio, Graph attention networks, arXiv preprint arXiv:1710.10903 (2017).

[27] J. Konc, D. Janežič, Probis algorithm for detection of structurally similar protein binding sites by local structural alignment, Bioinformatics 26 (9) (2010) 1160–1168 (2010).

[28] J. Fine, J. Konc, R. Samudrala, G. Chopra, CANDOCK: Chemical atomic network-based hierarchical flexible docking algorithm using generalized statistical potentials, Journal of chemical information and modeling 60 (3) (2020) 1509–1527 (2020).

[29] S. Lešnik, J. Konc, In silico laboratory: Tools for similarity-based drug discovery, in: Targeting Enzymes for Pharmaceutical Development, Springer, 2020, pp. 1–28 (2020).

[30] D. S. Johnson, M. A. Trick, Cliques, coloring, and satisfiability: second DIMACS implementation challenge, October 11-13, 1993, Vol. 26, American Mathematical Soc., 1996 (1996).

Dodatek A

Priloge

Tabela A.1: Tabela vsebuje napovedi parametra T_{limit} za posamezen model na testni množici naključnih grafov.

n	**p**	**MCQD**	**XGB**	**GCN**	**GAT**	**GIN**	**SVR-WL**
100	0,6	0,02500	0,00633	0,00987	0,01241	0,27739	0,00949
100	0,7	0,02500	0,01408	0,01067	0,01244	0,25116	0,02837
100	0,8	0,02500	0,02457	0,01259	0,01120	0,22769	0,00490
100	0,9	0,02500	0,02337	0,00995	0,01220	0,20211	0,00935
100	0,95	0,02500	0,01107	0,01037	0,01220	0,19147	0,01089
150	0,5	0,02500	0,00293	0,01140	0,01200	0,24161	0,00096
150	0,6	0,02500	0,00732	0,00945	0,01211	0,20812	0,00842
150	0,7	0,02500	0,02422	0,01242	0,01320	0,16441	0,00850
150	0,8	0,02500	1,00000	0,01233	0,01220	0,12980	0,11390
150	0,9	0,02500	0,95571	0,01111	0,01320	0,10144	0,00311
150	0,95	0,02500	0,33766	0,01015	0,01020	0,08866	0,00656
200	0,4	0,02500	0,00286	0,01076	0,01020	0,23031	0,00309
200	0,5	0,02500	0,00615	0,01716	0,01520	0,17661	0,01901
200	0,6	0,02500	0,64202	0,01195	0,01130	0,12809	0,00969
200	0,7	0,02500	0,62982	0,01317	0,01560	0,09296	0,00262
200	0,8	0,02500	1,00000	0,01178	0,01420	0,06053	0,00163

200	0,9	0,02500	0,98823	0,01240	0,01220	0,03850	0,00614
200	0,95	0,02500	0,43283	0,01043	0,01120	0,03036	0,01002
300	0,4	0,02500	0,24721	0,01431	0,01210	0,13172	0,09893
300	0,5	0,02500	1,00000	0,01658	0,01211	0,03773	0,01533
300	0,6	0,02500	0,95951	0,01280	0,01245	0,01755	0,01009
300	0,7	0,02500	0,28465	0,01268	0,01234	0,07750	0,01628
500	0,3	0,02500	0,54404	0,01342	0,01243	0,02251	0,00776
500	0,4	0,02500	0,53891	0,01368	0,01265	0,00474	0,00036
500	0,5	0,02500	0,58113	0,01371	0,01221	0,00076	0,00192
500	0,6	0,02500	0,25303	0,01670	0,01226	0,02204	0,00001
1000	0,2	0,02500	0,30768	0,01573	0,01222	0,00074	0,00014
1000	0,3	0,02500	0,83521	0,01364	0,01220	0,00001	0,00005
1000	0,4	0,02500	1,00000	0,01258	0,01222	0,00000	0,00008
1000	0,5	0,02500	1,00000	0,01258	0,01220	0,00000	0,00008

Tabela A.2: Tabela vsebuje napovedi parametra T_{limit} za posamezen model na testni množici naključnih gostih grafov.

n	**p**	**MCQD**	**XGB**	**GCN**	**GAT**	**GIN**	**SVR-WL**
63	0,9944	0,02500	0,00281	0,00939	0,01324	0,27161	0,00350
113	0,9987	0,02500	0,00259	0,00971	0,01223	0,14892	0,01015
121	0,9955	0,02500	0,00276	0,00989	0,01213	0,13194	0,03884
175	0,9954	0,02500	0,01128	0,01008	0,01265	0,04475	0,01343
304	0,9911	0,02500	0,01267	0,01038	0,01267	0,00072	0,00520
414	0,9943	0,02500	0,00694	0,01035	0,01298	0,00000	0,00286
443	0,9938	0,02500	0,01556	0,01049	0,01265	0,00000	0,00163
475	0,9979	0,02500	0,00218	0,01037	0,01228	0,00000	0,00058
476	0,9977	0,02500	0,00241	0,01036	0,01298	0,00000	0,00079
524	0,9992	0,02500	0,00254	0,01036	0,01325	0,00000	0,00181
622	0,9981	0,02500	0,00089	0,01046	0,01263	0,00000	0,00023
690	0,9978	0,02500	0,00462	0,01046	0,01265	0,00000	0,00050
828	0,9979	0,02500	0,00112	0,01052	0,01264	0,00000	0,00024
931	0,9995	0,02500	0,00220	0,01052	0,01265	0,00000	0,00007
941	0,9988	0,02500	0,00087	0,01063	0,01264	0,00000	0,00008

Tabela A.3: Tabela vsebuje napovedi parametra T_{limit} za posamezen model na testni množici majhnih proteinskih produktnih grafov.

n	**p**	**MCQD**	**XGB**	**GCN**	**GAT**	**GIN**	**SVR-WL**
61	0,9792	0,02500	0,00547	0,00961	0,01123	0,27834	0,00386
138	0,9422	0,02500	0,61101	0,01278	0,01325	0,11213	0,00447
200	0,8581	0,02500	0,62910	0,01218	0,01240	0,04731	0,00006
271	0,9852	0,02500	0,01352	0,01031	0,01431	0,00273	0,00723
346	0,9091	0,02500	0,45680	0,01273	0,01140	0,00042	0,00323
451	0,9743	0,02500	0,00991	0,01285	0,01222	0,00000	0,00208
563	0,9800	0,02500	0,02006	0,01152	0,01522	0,00000	0,00128
655	0,9692	0,02500	0,02717	0,01515	0,01453	0,00000	0,00029
750	0,9625	0,02500	0,16964	0,01346	0,01324	0,00000	0,00053
905	0,9412	0,02500	1,00000	0,01426	0,01123	0,00000	0,00077

Tabela A.4: Tabela vsebuje napovedi parametra T_{limit} za posamezen model na testni množici proteinskih produktnih grafov.

n	**p**	**MCQD**	**XGB**	**GCN**	**GAT**	**GIN**	**SVR-WL**
27840	0,0069	0,02500	0,30768	0,01235	0,01220	0,00587	0,00014
36841	0,0060	0,02500	0,83521	0,01234	0,01220	0,00417	0,00005
121359	0,0024	0,02500	1,00000	0,01313	0,01220	0,00020	0,00008

Tabela A.5: Tabela vsebuje napovedi parametra T_{limit} za posamezen model na testni množici proteinskih sidrnih grafov.

n	p	MCQD	XGB	GCN	GAT	GIN	SVR-WL
345	0,1266	0,02500	0,10253	0,01509	0,01233	0,35134	0,00078
1779	0,1108	0,02500	0,00559	0,01344	0,0012	0,24781	0,00000
1851	0,1580	0,02500	0,01068	0,01337	0,01220	0,16675	0,00000
3233	0,1620	0,02500	0,17193	0,01083	0,01520	0,00095	0,00000
4211	0,0448	0,02500	0,04081	0,01509	0,014220	0,35708	0,00000
5293	0,1119	0,02500	0,84119	0,01372	0,01243	0,19573	0,00000
5309	0,1474	0,02500	0,23140	0,01174	0,01320	0,00039	0,00000
5735	0,0592	0,02500	0,24322	0,01509	0,01120	0,34853	0,00000
6294	0,1382	0,02500	1,00000	0,01413	0,01520	0,00002	0,00000
7211	0,1012	0,02500	0,82883	0,01506	0,01222	0,30549	0,00000

Tabela A.6: Tabela vsebuje napovedi parametra T_{limit} za posamezen model na testni množici grafov DIMACS.

Ime grafa	MCQD	XGB	GCN	GAT	GIN	SVR-WL
MANN-a27	0,02500	0,01955	0,01048	0,01220	0,00003	0,00013
MANN-a45	0,02500	0,00343	0,01070	0,01220	0,00000	0,00001
MANN-a81	0,02500	0,48233	0,01079	0,01220	0,00000	0,00000
MANN-a9	0,02500	0,48233	0,00932	0,01220	0,31548	0,03469
brock200-1	0,02500	0,02790	0,01104	0,01220	0,07884	0,00667
brock200-2	0,02500	0,48233	0,01320	0,01220	0,17909	0,01373
brock200-3	0,02500	0,62982	0,01248	0,01220	0,12944	0,00523
brock200-4	0,02500	1,00000	0,01271	0,01220	0,10911	0,07743
brock400-1	0,02500	0,00069	0,01268	0,01220	0,00079	0,00495
brock400-2	0,02500	0,00091	0,01629	0,01220	0,00076	0,00252
brock400-3	0,02500	0,98823	0,01522	0,01220	0,00076	0,00216
brock400-4	0,02500	1,00000	0,01344	0,01220	0,00076	0,00025
brock800-1	0,02500	0,30501	0,01377	0,01220	0,00000	0,00051

brock800-2	0,02500	0,28447	0,01328	0,01220	0,00000	0,00517
brock800-3	0,02500	0,18021	0,01439	0,01220	0,00000	0,00025
brock800-4	0,02500	0,98823	0,01412	0,01220	0,00000	0,00114
c-fat200-1	0,02500	0,05655	0,01152	0,01220	0,34716	1,00000
c-fat200-2	0,02500	1,00000	0,01389	0,01220	0,32701	0,04864
c-fat200-5	0,02500	0,98823	0,02732	0,01220	0,21282	0,00157
c-fat500-1	0,02500	0,48233	0,01029	0,01220	0,34489	1,00000
c-fat500-10	0,02500	0,01232	0,03121	0,01220	0,03145	0,00200
c-fat500-2	0,02500	0,00909	0,01161	0,01220	0,31985	1,00000
c-fat500-5	0,02500	1,00000	0,01549	0,01220	0,19179	0,10256
hamming10-2	0,02500	1,00000	0,01087	0,01220	0,00000	0,00008
hamming10-4	0,02500	0,00606	0,02978	0,01220	0,00000	0,00762
hamming6-2	0,02500	0,09002	0,00920	0,01220	0,28817	0,00551
hamming6-4	0,02500	0,38601	0,00945	0,01220	0,34740	0,01374
hamming8-2	0,02500	0,38937	0,01049	0,01220	0,00609	0,00270
hamming8-4	0,02500	0,58458	0,01416	0,01220	0,08444	0,24279
johnson16-2-4	0,02500	0,28447	0,01150	0,01220	0,21606	0,02852
johnson32-2-4	0,02500	0,05001	0,01830	0,01220	0,00000	0,00465
johnson8-2-4	0,02500	1,00000	0,00898	0,01220	0,35035	0,01077
johnson8-4-4	0,02500	0,60597	0,00911	0,01220	0,30536	0,00883
keller4	0,02500	0,01264	0,01017	0,01220	0,17960	0,49997
keller5	0,02500	0,00955	0,01671	0,01220	0,00000	0,00021
keller6	0,02500	0,28447	0,02563	0,01220	0,00000	0,00000
p-hat1000-1	0,02500	0,01473	0,02562	0,01220	0,00423	0,00000
p-hat1000-2	0,02500	0,62982	0,03944	0,01220	0,00000	0,00012
p-hat1000-3	0,02500	0,05138	0,02363	0,01220	0,00000	0,00028
p-hat1500-1	0,02500	0,00615	0,03155	0,01220	0,00001	0,00001
p-hat1500-2	0,02500	0,00566	0,04633	0,01220	0,00000	0,00000
p-hat1500-3	0,02500	0,28447	0,02172	0,01220	0,00000	0,00000
p-hat300-1	0,02500	0,30501	0,02024	0,01220	0,23627	0,00448
p-hat300-2	0,02500	0,74546	0,02421	0,01220	0,07228	0,00153

p-hat300-3	0,02500	1,00000	0,01558	0,01220	0,01112	0,00398
p-hat500-1	0,02500	0,00353	0,02922	0,01220	0,10531	0,01289
p-hat500-2	0,02500	0,01405	0,03215	0,01220	0,00312	0,01057
p-hat500-3	0,02500	0,01293	0,01763	0,01220	0,00002	0,00254
p-hat700-1	0,02500	0,00469	0,02152	0,01220	0,03545	0,00764
p-hat700-2	0,02500	0,64202	0,02665	0,01220	0,00004	0,00060
p-hat700-3	0,02500	1,00000	0,01755	0,01220	0,00000	0,00002
san1000	0,02500	0,62982	0,01983	0,01220	0,00000	0,00000
san200-0,7-1	0,02500	0,23507	0,01151	0,01220	0,09396	0,00807
san200-0,7-2	0,02500	0,67263	0,01441	0,01220	0,08987	0,00911
san200-0,9-1	0,02500	1,00000	0,01184	0,01220	0,03861	0,00240
san200-0,9-2	0,02500	1,00000	0,01279	0,01220	0,03864	0,00510
san200-0,9-3	0,02500	0,62982	0,01089	0,01220	0,03873	0,01431
san400-0,5-1	0,02500	0,21420	0,01166	0,01220	0,02255	0,00071
san400-0,7-1	0,02500	1,00000	0,01828	0,01220	0,00164	0,00366
san400-0,7-2	,02500	0,85219	0,01522	0,01220	0,00166	0,00297
san400-0,7-3	0,02500	1,00000	0,01114	0,01220	0,00169	0,00278
san400-0,9-1	0,02500	0,32817	0,01199	0,01220	0,00005	0,00587
sanr200-0,7	0,02500	0,98823	0,01354	0,01220	0,09277	0,03641
sanr200-0,9	0,02500	1,00000	0,01249	0,01220	0,03902	0,00130
sanr400-0,5	0,02500	1,00000	0,01158	0,01220	0,02266	0,00084
sanr400-0,7	0,02500	0,01192	0,01187	0,01220	0,00167	0,00285

Tabela A.7: Tabela vsebuje napovedi parametra T_{limit} za posamezen model na testni množici uteženih proteinskih sidrnih grafov.

n	**p**	**MCQD**	**GAT**
345	0,1266	0,02500	0,02003
1779	0,1108	0,02500	0,01780
1851	0,1580	0,02500	0,01872
3233	0,1620	0,02500	0,02383
4211	0,0448	0,02500	0,01794
5293	0,1119	0,02500	0,01992
5309	0,1474	0,02500	0,01754
5735	0,0592	0,02500	0,01780
6294	0,1382	0,02500	0,02583
7211	0,1012	0,02500	0,01952