UNIVERZA NA PRIMORSKEM
FAKULTETA ZA MATEMATIKO, NARAVOSLOVJE IN INFORMACIJSKE TEHNOLOGIJE KOPER
Matematične znanosti,
magistrski študijski program (2. stopnja)

Kati Rozman

PROBLEM MAKSIMALNE KLIKE NA NEUSMERJENIH GRAFIH

Magistrska naloga

Mentorica:
Prof. dr. Dušanka Janežič

Koper, januar 2009

Osnovna literatura za študij maksimalne klike na neusmerjenih grafih in izdelavo magistrskega dela:

[20] Konc, J., Janežič, D., *An Improved Branch and Bound Algorithm for the Maximum clique problem*, MATCH Commun. Math. Comput. Chem., 2007, 58, 569-590.

IZJAVA

Izjavljam, da sem magistrsko delo izdelala samostojno pod mentorstvom prof. dr. Dušanke Janežič.

KATI ROZMAN

Zahvala

Zahvaljujem se mentorici prof. dr. Dušanki Janežič za vso strokovno pomoč pri nastajanju magistrske naloge. S svojim znanjem me je vseskozi spodbujala k novim spoznanjem, s strokovnim pregledom pa pripomogla k temu, da je magistrska naloga smiselno organizirana. Hvala, ker je bila vedno na voljo za moja vprašanja.

Posebna zahvala tudi dr. Janezu Koncu za vse intelektualne diskusije in pogovore in doc. dr. Klavdiji Kutnar za kritično branje magistrske naloge in za vse dobrodošle pripombe.

Čeprav je svet danes popolnoma drugačen od tistega, ki si ga poznala ti mami, je vez med nama ostala nespremenjena. Hvala ti, ker si do sedaj toliko let ustvarjala z mano.

Predvsem pa bi se za podporo v času nastajanja magistrske naloge rada zahvalila tudi sestri, Svetlani, Maji in Marku. Njihov prirojeni optimizem in dobrota sta mi pomagala, da sem verjela vase.

Povzetek

Magistrska naloga obravnava problem maksimalne klike na neusmerjenih grafih. Klika je podgraf grafa, v katerem so vsa vozlišča med seboj sosednja. Poiskati maksimalno kliko na neusmerjenem grafu, je eden od pomembnih NP-težkih problemov v diskretni matematiki in računalništvu. V nalogi najprej opišemo osnovne pojme iz teorije grafov, ki so potrebni za nadaljnje razumevanje. Posebno poglavje posvetimo problemu barvanja vozlišč grafa, kliki na neusmerjenem grafu in konstrukciji produktnih grafov. Največji skupni podgraf dveh grafov, ki ju primerjamo, je ekvivalenten maksimalni kliki v produktnem grafu. Sledi opis optimizacijskih problemov in algoritmov, s katerimi se lahko lotimo problema maksimalne klike. V zaključnem poglavju predstavimo MaxClique algoritem, ki poda približno rešitev za problem maksimalne klike na neusmerjenem grafu. Algoritem je prirejen za iskanje podobnosti v strukturah proteinov. Tako smo pristope teorije grafov uporabili za raziskovanje proteinskih struktur in napovedovanje vezavnih mest proteinov.

Ključne besede: graf, klika, maksimalna klika, NP-težek problem, MaxClique algoritem

Abstract

This master thesis considers with maximum cliques, complete subgraphs in which all pairs of vertices are adjacent. Finding a maximum clique in an undirected graph is one of the most important NP-hard problems in discrete mathematics and theoretical computer science. First some basics relating to the graph theory are given, which are necessary for the comprehension of this thesis. In the chapter that follows, vertex coloring problem and maximum clique problem of graphs are discussed and also the concepts of product graph construction are given. The product graph has the property that an maximum common subgraph between the graphs being compared is equivalent to a maximum clique in the product graph. Next, optimization problems and algorithms that can be used for solving maximum clique problem are described. In the final chapter, MaxClique algorithm, which gives an approximate solution to the maximum clique problem in an undirected graph, is presented. Algorithm is adapted for finding similarities in protein structures. The principles of graph theory are now being adopted to investigate protein structures and predicting protein binding sites.

Key words: graph, clique, maximum clique, NP-hard problem, MaxClique algorithm

Kazalo

Slike

Tabele

Uvod

Teorija grafov je eno od področij matematike, ki je v zadnjih desetletjih v velikem razvoju. Nedvomno gre to tudi na račun njene široke uporabnosti. Koncepti teorije grafov so uporabni za iskanje učinkovitih algoritmov v matematiki, bioinformatiki, računalništvu, iskanju optimalnih urnikov, za opis računalniških mrež, v lingvistiki, bančnih transakcijah, internetnih aplikacijah itd. Zelo široka je uporaba teorije grafov v naravoslovju, predvsem matematiki, bioinformatiki, molekularni biologiji, v bioloških sistemih, predvsem tam, kjer lahko strukture predstavimo kot grafe, neke množice točk in povezav ter med njimi iščemo podobnosti. Grafi so se izkazali kot odlično orodje za modeliranje problemov, ki jih srečamo tako v raziskovalnem kot tudi v vsakdanjem življenju.

V magistrski nalogi bomo predstavili področje maksimalne klike na neusmerjenih grafih, t.j. največje podstrukture v produktnem grafu. Opisana je tudi uporabnost maksimalne klike na področju naravoslovja, predvsem na področju molekularnega modeliranja. Najprej so predstavljene osnovne definicije. Podana je motivacija za študij produktnih grafov. Eno večjih in pomembnejših področij teorije grafov je barvanje grafov, kjer poskušamo grafe pobarvati s čim manj barvami, poleg tega pa upoštevamo še druge omejitve. Barvanje grafov pomeni dodelitev barve vsem točkam grafa, tako da dve po-

vezani točki nimata enake barve. Barvanje točk grafov bomo uporabili za določitev zgornje meje za maksimalno kliko na neusmerjenih grafih [20].

V nadaljevanju magistrske naloge se bomo omejili na opis problema, kako za dani neusmerjen graf poiskati kliko grafa z največjim številom vozlišč - maksimalno kliko. Polni graf K_n je graf z n vozlišči, v katerem obstaja povezava med vsakim parom različnih vozlišč. Maksimalna klika na neusmerjenem grafu je polni podgraf, ki ni vsebovan v nobenem drugem polnem podgrafu. Iskanje maksimalne klike na neusmerjenih grafih uvrščamo med NP-težke probleme (podobno kot barvanje točk, izomorfnost grafov, itd.). Problem je NP-težek, če zanj ne obstajajo učinkoviti polinomski algoritmi. Ko se torej v praksi srečamo s problemom, ki je NP-težek, je to zadosten razlog, da opustimo iskanje natančnega polinomskega algoritma in se raje posvetimo iskanju algoritmov, ki v korist hitrosti žrtvujejo natančnost. Z algoritmi v teoriji grafov lahko natančno opišemo, kako se določen problem rešuje. Učinkovitost algoritma oziroma njegova časovna zahtevnost je lahko izmerjena kot število osnovnih korakov, ki jih je treba izvesti za rešitev problema.

V literaturi tako najdemo mnoge različice algoritmov in hevristike za reševanje NP-težkih problemov. V delu so predstavljeni trije algoritmi za iskanje maksimalne klike na neusmerjenih grafih. Bron-Kerbosch algoritem [6], Tomita-Seki algoritem [31] in MaxClique algoritem [20]. Slednji v manj korakih in krajšem času kot prva dva pripelje do dokaj natančnih rezultatov rešitve zastavljenega problema iskanja maksimalne klike na neusmerjenem grafu.

Leta 1973 sta Bron in Kerbosch [6] predstavila algoritem za iskanje vseh klik na neusmerjenem grafu z uporabo "razveji in omeji" metode. To je metoda za iskanje rešitev problema, pri kateri hranimo možne rešitve in ocenjujemo njihovo obetavnost. Kasneje je Tomita s sodelavci [31] predstavil algoritem za približno barvanje grafov, ki zagotavlja barvanje vozlišč v osnovnem algoritmu. Za MaxClique algoritem [20] pa je koristno poznati čimbolj natančne meje, zato je ideja algoritma za izračun maksimalne klike na neusmerjenih

grafih, ki so ga razvili na Kemijskem inštitutu v Ljubljani ta, da bi bile tesne zgornje meje izračunane že v začetku algoritma, saj se s tem močno skrajša čas, ki je potreben za rešitev problema.

Naštete algoritme se veliko uporablja na področju naravoslovja, zlasti molekularnega modeliranja, kjer iščejo vezavna mesta proteinov s pomočjo matematične teorije grafov, t.i. proteinskih grafov in maksimalnih klik na produktnih grafih proteinskih grafov. Proteinski graf je definiran s pomočjo tridimenzionalne strukture proteina, na katerem želimo napovedati vezavna mesta (t.j. najbolj ohranjen del površine proteina). Izbran protein s pristopi teorije grafov pretvorimo v proteinski graf, tako da vsaka funkcionalna skupina točk na površini proteina predstavlja vozlišče, vezi pa predstavljajo povezave med tistimi vozlišči, ki so med seboj povezana. Dobljeni proteinski graf primerjamo z vsakim proteinskim grafom njegovih strukturnih sosedov. V proteinskem grafu vsakemu vozlišču priredimo matriko razdalj, le-te pa uporabimo za primerjavo. Med dvema proteinskima grafoma konstruiramo produktni graf, največje število ujemajočih se vozlišč grafa pa predstavlja maksimalno kliko na neusmerjenem grafu. Največja podobnost med dvema proteinoma, ki služi za določitev najbolj ohranjene površine proteina, je maksimalna klika.

Poglavje 1

Teorija grafov

1.1 Osnovne definicije

Naj bo $V = \{1, 2, ..., n\}$ končna neprazna množica in E poljubna družina dvoelementnih podmnožic množice V. Paru $G = (V, E)$ pravimo ***neusmerjen graf*** G na množici *vozlišč (točk)* $V = V(G)$ in z množico *povezav* $E = E(G)$. Povezave so neurejeni pari krajišč, kar pomeni, da zapis $\{u, v\}$ pomeni isto kot $\{v, u\}$. Element $\{u, v\}$ množice E pišemo krajše kot uv. Ko imamo urejene pare vozlišč, govorimo o ***usmerjenih grafih*** ali ***digrafih***. Tem urejenim parom vozlišč rečemo *loki* ali *usmerjene* povezave in jih označimo z (u, v). Lok $e = (u, v)$ je usmerjen od vozliščaa u proti vozlišču v. V tem primeru rečemo, da je u *rep* loka e, v pa *glava* loka e.

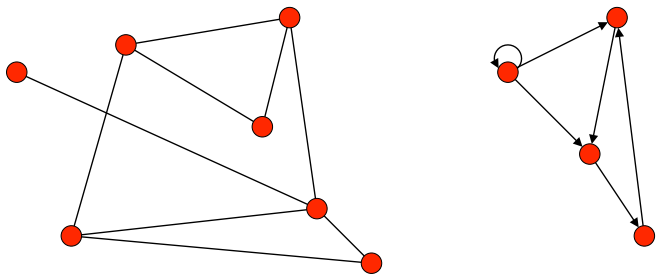
Včasih dopuščamo tudi grafe, ki imajo med nekaterimi pari vozlišč več povezav (*vzporedne povezave*) ali pa imajo povezave, ki imajo obe krajišči enaki (*zanke*). Takim grafom pravimo ***multigrafi***. Če graf ne vsebuje zank in

vzporednih povezav, ga imenujemo ***enostaven*** graf. Slika 1.1 prikazuje primer neusmerjenega grafa in usmerjenega multigrafa.

Kadar je par vozlišč uv element množice E, pravimo, da sta vozlišči u in v ***sosednji*** v grafu G. Povezavi e_1 in e_2, $e_1 \neq e_2$ sta ***sosednji***, če imata kako skupno krajišče, v nasprotnem primeru sta ***neodvisni***. Torej za vozlišče $u \in V$, je množica $\Gamma(u)$ množica vseh vozlišč $v \in V$, ki so sosednja vozlišču u. Sosednost paroma povezanih vozlišč nam definira ***matrika sosednosti*** $A = (A_{ij})_{n \times n}$ grafa G, kjer sta v_1 in v_2 vozlišči grafa G.

$$A_{ij} := \begin{cases} 1, & \text{če} \quad (v_i, v_j) \in E \\ 0, & \text{sicer} \end{cases}$$

Število sosedov $\Gamma(v_i)$ vozlišča v_i je tesno povezano z stopnjo vozlišča v_i. V neusmerjenem grafu G je ***stopnja*** ali ***valenca*** vozlišča u enaka številu povezav $|\Gamma(u)| = \{v|(uv) \in E\}$ grafa G, ki imajo vozlišče u za svoje krajišče. Označimo jo z $|\Gamma(u)|$, najmanjšo stopnjo vozlišča grafa označimo z $\delta(G)$, največjo pa z $\Delta(G)$.

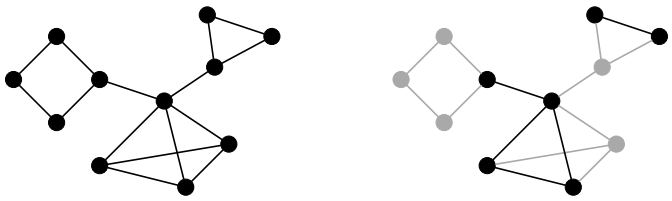


Slika 1.1: Neusmerjen graf in usmerjen multigraf

Pri delu z grafi nas pogosto ne zanima celoten graf ampak le njegov del, saj je celoten graf lahko prevelik ali pa opazujemo del zato, ker ima neko posebno lastnost.

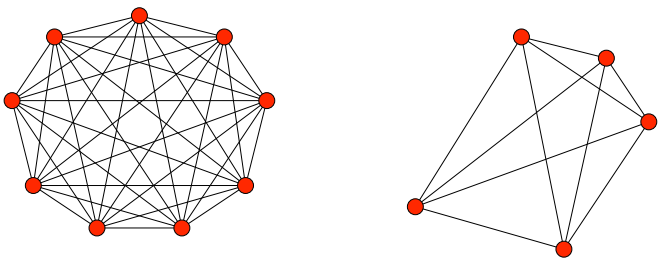
Graf H je ***podgraf*** grafa G in G je ***nadgraf*** grafa H, če je $V(H) \subseteq V(G)$ in $E(H) \subseteq E(G)$. Če je $F \subseteq E(G)$, potem z $G[F]$ označimo ***vpet*** podgraf grafa G, ki je določen s povezavami iz F. Zanj velja $V(G(F)) = V(G)$ in $E(G(F)) = F$. Vpeti podgrafi vsebujejo vsa vozlišča originalnega grafa. Za povezave to ne velja nujno. Slika 1.2 prikazuje primer podgrafa.

Inducirani podgrafi grafa G so določeni s podmnožicami vozlišč grafa G. Če je $U \subseteq V(G)$, potem je $G(U)$ podgraf grafa G induciran z množico U. Zanj velja $V(G(U)) = U$ in $E(G(U)) = \{uv \in E(G) \mid u, v \in U\}$. Zapišemo lahko tudi $G(U) = (U, E \cap U \times U)$. Inducirani podgrafi ne vsebujejo nujno vseh vozlišč in povezav grafa G.



Slika 1.2: Primer grafa in podgrafa

Polni graf reda n je graf z n vozlišči in vsemi možnimi povezavami. Teh je $\frac{n(n-1)}{2}$. Označimo ga s K_n. Primer je prikazan na sliki 1.3, kjer si lahko ogledamo tudi primer induciranega podgrafa.

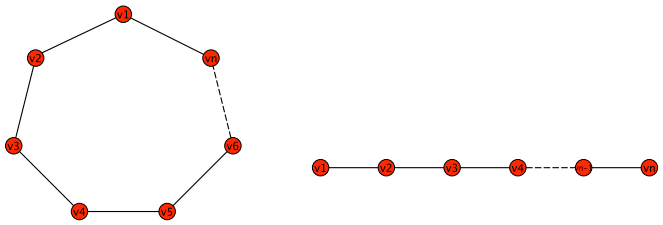


Slika 1.3: Polni graf K_9 in induciran podgraf grafa K_9

Naj bo G graf. Njegov ***komplement*** G^C je graf z vozlišči $V(G^C) = V(G)$, s povezavami $E(G^C) = \{e; e \notin E(G)\}$. V komplementu so torej natanko tiste povezave, ki jih ni v originalnem grafu.

Sprehod v grafu $G = (V, E)$ je tako zaporedje vozlišč $u_1, u_2, u_3, ..., u_n$, da je $u_i u_{i+1} \in E$ za vsak $i \in \{1, 2, \ldots, n-1\}$. Vozlišče u_1 imenujemo začetek in u_n konec sprehoda. Če je $u_n = u_1$, sprehodu pravimo ***obhod***.

Če so v sprehodu sama različna vozlišča, dobimo podgraf, ki ga imenujemo ***pot***. Pot pogosto na kratko zapišemo kar z zaporedjem vozlišč $v_1 v_2 ... v_n$ in jo označimo z P_n.



Slika 1.4: CIKEL C_7 IN POT P_6 GRAFA

Obhod s samimi različnimi vozlišči je ***cikel***. Pravimo mu tudi *n-cikel* in ga označimo s C_n. Krajši zapis je spet kar zaporedje vozlišč s ponovljenim začetnim vozliščem $v_1 v_2 ... v_n v_1$. Cikel dobimo, če v poti povežemo prvo in zadnje vozlišče. Glej sliko 1.4. Cikel je ***hamiltonski***, če gre skozi vsako vozlišče grafa natanko enkrat. Graf je ***hamiltonski***, če vsebuje hamiltonski cikel. Posebno družino hamiltonskih grafov so predkratkim uspešno uporabili za definiranje vzporednih računalniških povezav za simulacije molekulske dinamike [23].

Graf G je ***povezan***, če med poljubnima vozliščema $u, v \in V$ obstaja vsaj en sprehod, sicer je nepovezan. Vsak graf ima očitno vsaj eno komponento

in vsak nepovezan graf ima vsaj dve komponenti. Povezanost grafa nas zanima, ker lahko pri reševanju problemov rešitev pogosto najdemo za vsako komponento posebej in jo nato združimo za celoten graf.

Razdalja med vozliščema u in v je dolžina najkrajše poti med njima in jo označimo z $d(u,v)$ ali $diam(u,v)$. Tako rečemo, da je vozlišče u za $d(u,v)$ oddaljeno od vozlišča v. Največjo razdaljo med vozliščema v grafu imenujemo *premer* ali *diameter* grafa G.

Razdaljna matrika je definirana kot simetrična matrika $D = (D_{u,v})_{n\times n}$ grafa G, kjer je $d(u,v)$ dolžina najkrajše poti med vozliščema u in v v grafu G.

$$D_{uv} := \begin{cases} d(u,v), & \text{če} \quad (i \neq j) \\ 0, & \text{sicer} \end{cases}$$

Graf G je ***r-delen***, če lahko množico $V(G)$ razbijemo na r disjunktnih podmnožic, tako da ima vsaka povezava $e \in E(G)$ krajišči v različnih podmnožicah. Če je $r = 2$, pravimo grafu G dvodelen graf. Graf je dvodelni natanko tedaj, ko ne vsebuje lihih ciklov.

Drevo je povezan graf brez cikla. To pomeni, da ne obstaja podmnožica vozlišč grafa, katere induciran graf je cikel. Grafu, ki je sestavljen iz enega ali več dreves, pravimo tudi ***gozd***.

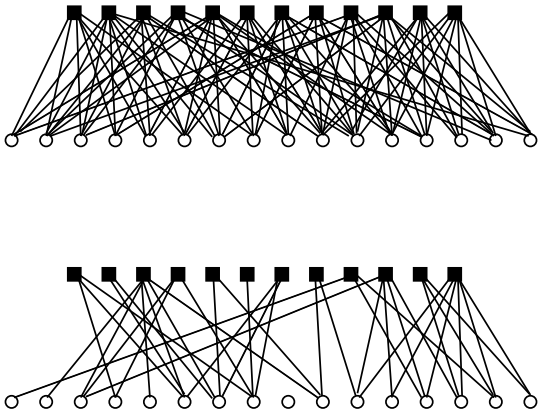
Homomorfizem iz grafa G v graf H je preslikava $f : V(G) \longrightarrow V(H)$, za katero je $f(u)f(v) \in E(H)$, če je $uv \in E(G)$. Tak homomorfizem f označimo z $f : G \longrightarrow H$. Če obstaja homomorfizem $f : G \longrightarrow H$, pišemo $G \longrightarrow H$, oziroma $G \nrightarrow H$, če tak homomorfizem ne obstaja. Homomorfizmi so torej preslikave vozlišč, ki ohranjajo sosednost. Rečemo lahko, da je G ***homomorfen*** grafu H, oz., da je graf G ***H-obarvljiv***. Kompozitum $f \circ g$ homomorfizmov $g : G \longrightarrow H$ in $f : H \longrightarrow K$ je homomorfizem iz G v K.

Izomorfizem je bijektivni homomorfizem. ***Avtomorfizem*** je izomorfizem grafa nase. Grupa ***Aut(G)*** simetrij grafa $G = (V, E)$ vsebuje natanko tiste bijektivne preslikave na množici V, ki ohranjajo sosednost.

Definirajmo sedaj še gostoto grafov. ***Gostota grafa*** D za neusmerjen graf $G = (V, E)$ je definirana kot

$$D = \frac{|E|}{|V| \cdot \frac{|V|-1}{2}}.$$

Definicija velja le, če velja $|V| > 1$, v nasprotnem primeru je $D = 0$. Neusmerjen graf G ima največ $|V| \cdot \frac{|V|-1}{2}$ povezav in zato je največja možna gostota grafa enaka 1. Graf G je ***redek***, ko je $|E| \approx |V|$. Graf G je ***gost***, če je število povezav grafa $|E| \approx |V^2|$. Pogosto grafom pravimo gosti, če velja $D > 0,5$. Spodnja slika 1.5 prikazuje primer gostega grafa in primer redkega grafa. Razlika med številom povezav v gostih in redkih grafih je očitna.



Slika 1.5: Gostota grafov

Neusmerjeni grafi bodo skozi delo poimenovani grafi. Vsi v nadaljevanju omenjeni grafi bodo končni. Prav tako bodo naši grafi enostavni in povezani. Velik pomen pa bomo pripisali gostoti grafov.

Poglavje 2

Klika neusmerjenega grafa

2.1 Barvanje grafov

Za boljšo predstavo o homomorfizmih si poglejmo povezavo z barvanjem vozlišč v grafu. Definirajmo ***k-barvanje*** grafa G kot dodelitev k barv vozliščem grafa G, tako da sta sosednji vozlišči obarvani različno. Vsakemu vozlišču dodelimo natanko eno barvo.

Označimo s K_k poln graf na vozliščih $1, 2, ..., k$ in označimo z istimi števili tudi barve v k-barvanju. V tem primeru lahko k-barvanje grafa G gledamo kot preslikavo $f : V(G) \longrightarrow \{1, 2, ..., k\}$. Zahteva, da so sosednja vozlišča različno obarvane, pomeni, da je $f(u) \neq f(v)$, če je $uv \in E(G)$. Pogoj $f(u) \neq f(v)$ je ekvivalenten pogoju $f(u)f(v) \in E(K_k)$ [26].

Definicija 2.1.1 ***Kromatično število*** $\chi(G)$ *grafa* G*, je definirano kot najmanjše naravno število* k*, za katero obstaja* k*-barvanje grafa* G*. Grafu s* $\chi(G) = k$ *rečemo* ***k-kromatičen*** *graf.*

Pridemo do naslednje trditve:

Trditev 2.1.2 *Homomorfizmi $f : G \longrightarrow K_k$ so natanko k-barvanja grafa G.*

Ta trditev nam omogoča, da izpeljemo naslednjo posledico, ki govori o kromatičnih številih.

Posledica 2.1.3 *Če je $G \longrightarrow H$, potem je $\chi(G) \leq \chi(H)$.*

Dokaz: Naj bo $h : G \longrightarrow H$ homomorfizem. Če za graf H obstaja k-barvanje $f : H \longrightarrow K_k$, potem je $f \circ g$ k-barvanje grafa G. Torej za graf G obstaja $\chi(H)$-barvanje in velja $\chi(G) \leq \chi(H)$.

Pri barvanju grafov veljajo naslednje lastnosti:

1. $\chi(G) \leq |V(G)|$ in enakost velja natanko takrat, ko je G poln graf;
2. če je H podgraf v G, potem je $\chi(H) \leq \chi(G)$;
3. če G vsebuje kliko na k točkah, potem je $\chi(G) \geq k$ *(Več o klikah v nadaljevanju dela)*.

Zgornja meje za $\chi(G)$ grafa G z n vozlišči je seveda $\chi(G) \leq n$. Vendar je ta meja običajno zelo slaba. Izboljšamo jo, če poznamo največjo stopnjo vozlišča grafa. Poglejmo si zvezo med Δ in χ.

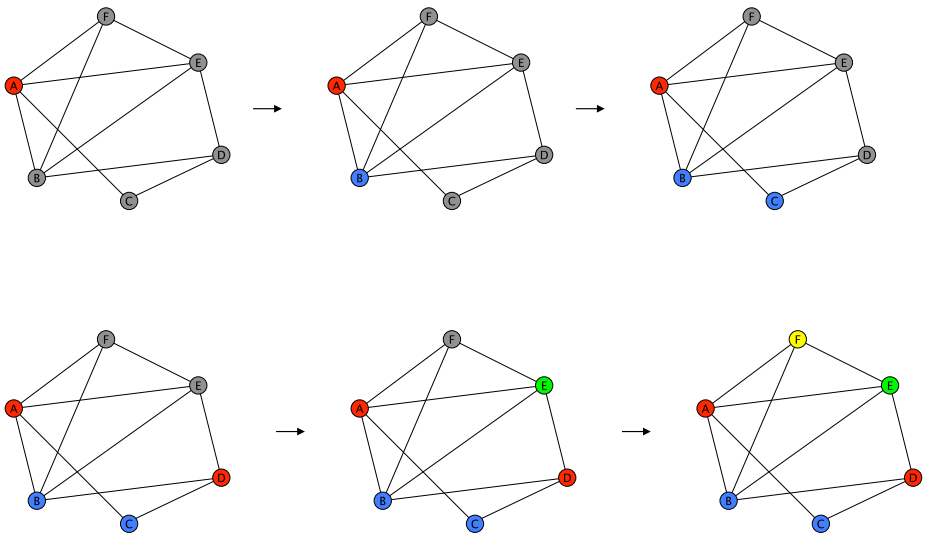
Trditev 2.1.4 *Za vsak graf G velja $\chi(G) \leq \Delta + 1$.*

Dokaz: Indukcija po številu vozlišč n grafa G. Očitno velja za $n = 1$. Naj bo $n \geq 2$ in naj bo v poljubno vozlišče v G. Ker je $\Delta(G - v) \leq \Delta$, obstaja barvanje c grafa $G - v$ z $\Delta + 1$ barvami. Vozlišče v ima največ Δ sosedov in vse ta vozlišča so že pobarvana. Ker imamo $\Delta + 1$ barv, je vsaj ena neuporabljena oz. prosta pri sosedih vozlišča v. Uporabimo to barvo na v in tako razširimo barvanje c na dobro barvanje grafa G.

□

Graf ni težko pobarvati, težje je najti najmanjše število barv, s katerimi ga lahko pobarvamo, t.i. kromatično število grafa. Poljuben tip grafa pa lahko v doglednem času pobarvamo le, če ima graf majhno število vozlišč. V kolikor so naš izbor grafi, ki premorejo veliko število vozlišč, se poslužujemo drugih metod [30].

Ena izmed njih je ***požrešna metoda*** barvanja vozlišč. S požrešno metodo skušamo postopoma zgraditi rešitev tako, da na vsakem koraku izberemo najboljšo možnost, ki v tem koraku največ doprinese h končnemu cilju in hkrati še vedno tvori dopustno rešitev. Je enostavna metoda, ki ponavadi vodi v učinkovit algoritem. S požrešno metodo iščemo najboljše rešitve problemov in jo uporabljamo v primerih, ko iščemo rešitev, ki najbolje ustreza določenemu kriteriju: npr. najkrajši čas, najkrajša pot, itd. Na tekočem koraku poiščemo element, ki bi prinesel največ h kriterijski funkciji. Ta element sprejmemo le, če s tem naša podmnožica še vedno zadošča zahtevam.



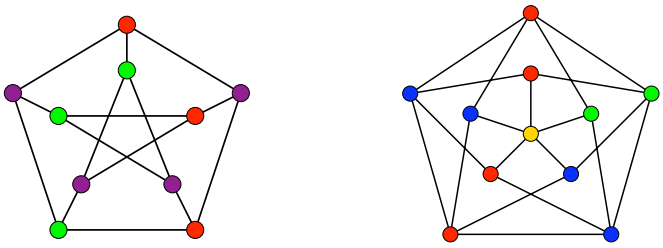
Slika 2.1: Določitev zgornje meje

Določitev zgornje meje, s pomočjo barvanja nam opiše postopek na sliki 2.1. Za dani graf $G = (V, E)$ pobarvaj vozlišče v_i z barvo c_i. Za vsako ustrezno vozlišče v_i izberi najmanjšo možno barvo tako, da je barvanje pravilno [29].

Primeri kromatičnih števil za grafe:

1. $\chi(K_n) = n$
2. $\chi(C_{2n}) = 2$
3. $\chi(C_{2n+1}) = 3$

Slika 2.2 prikazuje dobro barvanje Petersenovega in Grötzchevega grafa. Kromatično število prvega je 3, drugega pa 4.



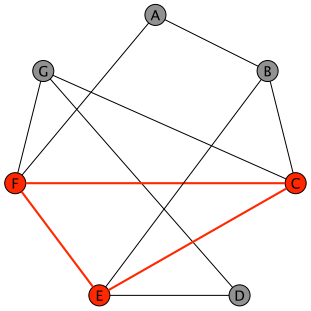
Slika 2.2: Kromatično število grafa

Obstaja tudi metoda za določitev spodnje meje za $\chi(G)$, pri kateri poiščemo največji polni podgraf grafa G. Več v nadaljevanju.

2.2 Maksimalna klika grafa

Definicija 2.2.1 *Podan je graf $G = (V, E)$ in podmnožico vozlišč $Q \subseteq V$. Inducirani podgraf grafa $G = (V, E)$, ki je izomorfen polnemu grafu, imenujemo* ***klika Q*** *grafa G.*

Z drugimi besedami, v kliki na neusmerjenem grafu so vsa vozlišča sosednja. Kot primer si poglejmo graf G na sliki 2.3, kjer vozlišča C, E in F tvorijo kliko Q velikosti 3.

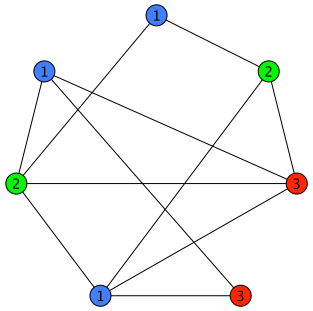


Slika 2.3: Klika Q na grafu G

Definicija 2.2.2 *Klika Q na neusmerjenem grafu G je* ***največja*** *(maximal), če ne obstaja klika D na grafu G, tako da $Q \subseteq D$ in $Q \neq D$. Kliko Q_{max} imenujemo* ***maksimalna*** *(maximum) klika, če ne obstaja klika na grafu G, ki bi vsebovala več vozlišč kot največja klika Q. Število vozlišč v maksimalni kliki označimo z $\omega(G)$ in ga imenujemo* ***klično število****.*

Definicija 2.2.3 *Množica $S \subseteq V(G)$ je* ***neodvisna množica*** *vozlišč grafa G, če noben par vozlišč iz množice S ni soseden v grafu G. Velikost največje neodvisne množice grafa G imenujemo* ***neodvisnostno število*** *grafa G in ga označimo z $\alpha(G)$.*

Definicija maksimalne klike je v tesni povezavi z definicijo maksimalne neodvisne množice. Neodvisna množica grafa G je ekvivalentna maksimalni kliki v komplementu grafa G^C.



Slika 2.4: Pravilno barvanje

Metoda barvanja vozlišč grafa lahko v veliki meri pripomore pri iskanju klik Q na neusmerjenem grafu G. Najmanjše število uporabljenih barv pri barvanju vozlišč grafa označuje že poznano kromatično število $\chi(G)$. V splošnem velja, $\omega(G) \leq \chi(G)$. Vsako pravilno barvanje $\chi(G)$ je torej zgornja meja za $\omega(G)$. Za primer vzemimo graf na sliki 2.4. Vozliščem klike, ki jo predstavlja trikotnik CEF iz slike 2.3 so dodeljene 3 različne barve.

Klično število $\omega(G)$ grafa G je za grafe, ki nimajo posebnih lastnosti

$$\omega(G) \geq \sum_{i=1}^{n} \frac{1}{n - d_i},$$

kjer je d_i stopnja vozlišča i v grafu G. Primeri kličnih števil za grafe:

1. $\omega(K_n) = n$
2. $\omega(C_{2n}) = 2$
3. $\omega(C_{2n+1}) = 3$

4. Grötzchev graf ($\omega = 2$)

5. Petersenov graf ($\omega = 2$)

Grafu, v katerem imajo vsa vozlišča enako stopnjo pravimo *regularen* graf in če je ta stopnja enaka k, rečemo, da je graf k-regularen. Graf G je k-povezan ($k \in \mathbb{N}$), če ima G vsaj $k+1$ vozlišč in je $G - X$ povezan za vsako podmnožico vozlišč $X \subset V(G)$, za katero je $|X| < k$. Največji k, za katerega je graf G k-povezan, je povezanost grafa. Graf G je povezavno k-povezan, če ima G vsaj $k+1$ povezav in je $G - X$ povezan za vsako množico povezav $X \subset E(G)$, za katero je $|X| < k$.

Za kliko Q velikosti k velja $\delta(G(Q)) = \Delta(G(Q)) = k - 1$. Stopnja vozlišča v kliki ne more biti večja. Klike so torej $(k-1)$-regularne oz. pravimo jim tudi *popolno goste*. Klike so *popolno kompaktne*. Kar pomeni, da je razdalja med poljubnima vozliščema klike enaka $diam(G(U)) = 1$. Klike so tudi *popolno povezane*. Klika Q velikosti k je $(k-1)$-vozliščno povezana in $(k-1)$-povezavno povezana. Osnovne definicije o razdalji in premeru grafa so navedene v poglavju 1.

Definicija 2.2.4 *Naj bo $G = (V, E)$ neusmerjen graf in $Q \subseteq V$ in $k > 0$, $k \in \mathbb{N}$. Potem je Q k-klika, natanko tedaj, ko za vsa vozlišča $u, v \in Q$ velja $d_G(u, v) \leq k$.*

Če je Q klika na neusmerjenem grafu in $v \in Q$, potem je tudi $Q - \{v\}$ klika grafa. Vsaka klika velikosti k vsebuje kliko velikosti $k-1$, celo več klik velikosti $k-1$.

Lema 2.2.5 *Naj bo $Q \subseteq V$ klika na neusmerjenem grafu G in $B \subseteq Q$. Potem je množica B tudi klika grafa G.*

Dokaz: Naj bo $Q \subseteq V$ klika na neusmerjenem grafu G in naj velja $B \subseteq Q$. Po defininiciji 2.2.2 in 2.2.8 velja, da vsak induciran podgraf polnega grafa tvori kliko. Iz tega sledi, da je tudi $B \subseteq V$ klika grafa G. □

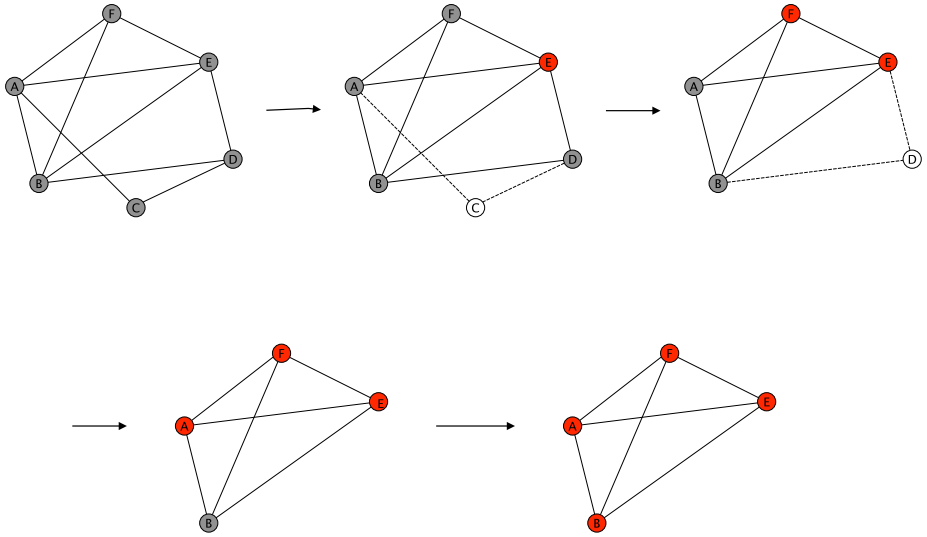
Spomnimo se, da je za $v \in V$ množica $\Gamma(v) = \{u \in V | uv \in E\}$ množica sosedov vozlišča v. Stopnja vozlišča v je $\delta(v) = |\Gamma(v)|$. Množica vozlišč $Q \subseteq V$ predstavlja graf $G_Q = (Q, E \cap (Q \times Q))$ induciran na množici vozlišč Q. $\Gamma_Q(v) = \{u \in Q | uv \in (Q \times Q) \cap E\}$ so sosedi vozlišča v v induciranem grafu na Q s stopnjo $\delta_Q(v) = |\Gamma_Q(v)|$. Množica $P = \cap_{v \in Q} \Gamma(v)$ predstavlja množico kandidatov za ***razširitev klike*** Q. Največja klika lahko vsebuje največ $|Q| + |P|$ vozlišč. Klika $\tilde{Q}$ je razširitev klike Q, če velja $Q \subseteq \tilde{Q}$. Razširitve bomo označevali z $\tilde{Q}(Q)$.

Podgraf $\mathcal{Z}(G) = \{I \subseteq V |\ G_I$ je povezana komponenta v G$\}$ vsebuje množico vozlišč povezane komponente v podanem neusmerjenem grafu G.

Poglejmo si postopek za razširitev klike $Q \subseteq V$ neusmerjenega grafa G v večjo kliko, ki ga uporabljamo za določitev spodnjih mej pri velikosti maksimalne klike [29]:

1. Podan je graf $G = (V, E)$;
2. Naj bo $Q = \emptyset$;
3. Naj bo $v \in G$ in $v \notin Q$ vozlišče z največjo stopnjo. Dodaj v v množico Q.
4. Izbriši vsa vozlišča iz V, ki niso sosednja vozlišču v.
5. Če je V prazna, končaj postopek. V nasprotnem primeru se vrni na korak 3.

Klika Q na neusmerjenem grafu je lahko razširjena v večjo kliko, če obstaja vozlišče grafa, ki ni v obstoječi kliki in je sosednje z vsemi vozlišči klike. Razširitev klike Q v večjo kliko je prikazana na sliki 2.5. Vsi kandidati nove množice $Q = \{E, F, A, B\}$ so obarvani rdeče. Vsebuje vsa tista vozlišča $w \in P$, ki so sosednja prvotno izbranemu vozlišču E z največjo stopnjo.



Slika 2.5: Ilustracija požrešne metode, $\omega(G) \geq 4$

Pokazali bomo dve enostavni lemi, ki pomagata omejiti množico kandidatov P in posledično zmanjšajo izbor tistih vozlišč, ki lahko služijo razširitvi klike Q. Lemi sta povzeti po [13].

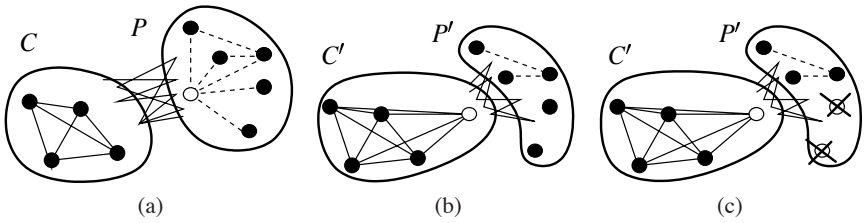
Prva lema govori o tistih vozliščih v množici P, ki ne bodo nikoli služila za razširitev izbrane klike Q v maksimalno kliko Q_{max} na grafu G.

Lema 2.2.6 *Podan je neusmerjen graf G, klika Q na grafu G in množica kandidatov $P = \cap_{v \in Q} \Gamma(v)$. Še več, naj bo $\sigma \in \mathbb{N}$ spodnja meja za velikost maksimalne klike Q_{max} na neusmerjenem grafu G. Potem vozlišče $v \in P$ za katerega velja $|Q| + \delta_P < \sigma - 1$ ne more razširiti klike Q v maksimalno kliko Q_{max} na grafu G.*

Dokaz: Predpostavimo, da P vsebuje vozlišče $v \in P$, za katerega velja $|Q| + \delta_P < \sigma - 1$. Vozlišče v pa razširi kliko Q v maksimalno kliko $Q^*(Q)$ z $|Q^*(Q)| = k$ vozlišči. Potem za vsak $u \in Q^*(Q)$ stopnja enaka $\delta_{Q^*}(u) =$

$k-1$. Ker je σ spodnja meja za velikost maksimalne klike $Q^*(Q)$ potem velja $\sigma - 1 \leq k - 1 = \delta_{Q^*(Q)}(v)$. Stopnjo $Q^*(Q)$ lahko razstavimo na particijo $\delta_{Q^*(Q)}(v) = \delta_Q(v) + \delta_{Q^*\setminus Q)}(v)$. Dobimo $\sigma - 1 \leq k - 1 = \delta_{Q^*(Q)}(v) = \delta_Q(v) + \delta_{Q^*\setminus Q)}(v) \leq |Q| - 1 + \delta_P(v) < \sigma - 1$, kar vodi v protislovje.

□



Slika 2.6: Shematičen prikaz razširitve klike Q

Druga lema določi vsa tista vozlišča, ki bodo vsebovana v vsaki razširitvi trenutne klike v maksimalno kliko.

Lema 2.2.7 *Podan je neusmerjen graf G, klika Q na grafu G in množica kandidatov $P = \cap_{v \in Q}\Gamma(v)$. Potem je vsako vozlišče $v \in P$, za katerega velja $\delta_P(v) = |P| - 1$, vsebovano v vsaki maksimalni kliki grafa G, ki vsebuje tudi kliko Q.*

Dokaz: Naj bo $Q^*(Q)$ maksimalna klika in $Q \subseteq Q^*$. Potem velja $Q^* \subseteq Q \cup P$ (po definiciji množice P). Predpostavimo, da za $v \in P$ velja $\delta_P(v) = |P| - 1$, $v \notin Q^*$. Za vozlišče v vemo, da je sosednje z vsemi vozlišči v Q (ker je vsebovano v množici P) in sosednje tudi z vsemi vozlišči v množici P (stopnje vozlišč v P). Torej iz tega sledi, da je vozlišče v sosednje tudi z vsemi vozlišči $u \in Q^* \subseteq Q \cup P$, kar pomeni, da je $Q^* \cup \{v\}$ klika vsebovana v Q in večja od klike Q^*. To pa je v nasprotju z predpostavko zgoraj, da je Q^* maksimalna klika. □

Slika 2.6 nam prikazuje trenutno kliko Q in množico kandidatov za razširitev P (a), po upoštevani lemi 2.2.7, ko dodamo vozlišče, dobimo razširjeno kliko (b), rezultat sta vozlišči s stopnjo 0 in ju lahko odstranimo iz postopka za razširitev, lema 2.2.6 (c).

Lema 2.2.8 *Vsaka maksimalna klika $Q_{max} \subseteq V$ na neusmerjenem grafu G je največja klika.*

Dokaz: Naj bo $Q_{max} \subseteq V$ maksimalna klika na neusmerjenem grafu G. Predpostavimo, da klika Q_{max} ni največja klika. Potem po definiciji 2.2.2 obstaja klika $D \subseteq V$, tako da $Q_{max} \subseteq D$ in $Q_{max} \neq D$. Zato velja $D - Q_{max} \neq \emptyset$ in klika D premore več vozlišč kot klika Q_{max}, kar pa nasprotuje predpostavki, da je Q_{max} maksimalna klika. Torej drži, da je Q_{max} maksimalna klika na neusmerjenem grafu G.

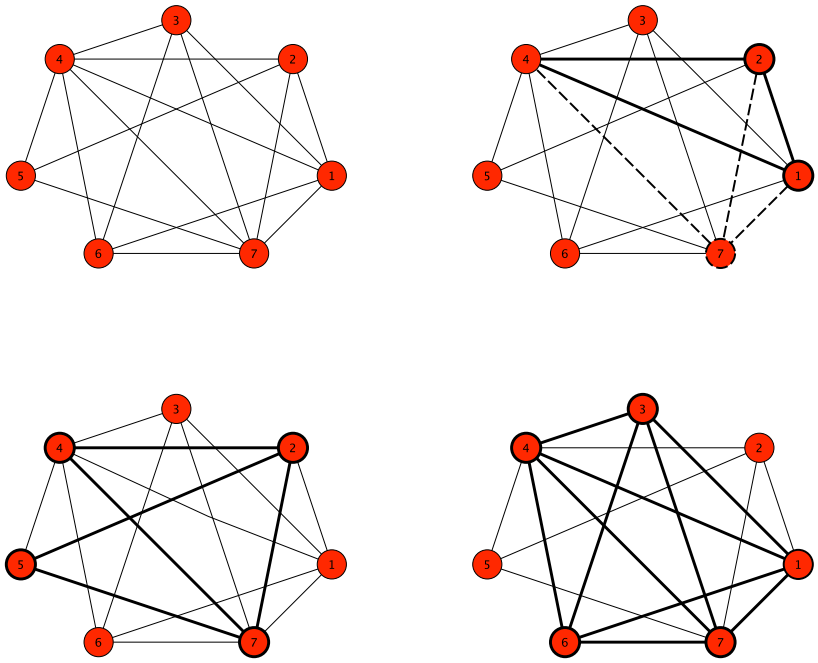
□

Največje klike danega neusmerjenega grafa so tiste klike Q, za katere je množica kandidatov iz P, s katerimi bi jih lahko razširili, prazna. Omenimo še množico vozlišč $S \subseteq V - Q - P$, s katerimi ne moremo razširiti klike Q v večjo kliko. Tista vozlišča, ki jih pri trenutni razširitvi ne upoštevamo premaknemo v množico S. Množica S pa nam omogoči, da se izognemo tistim klikam, ki smo jih že določili.

Primer 2.2.9 *Neusmerjen graf na sliki 2.7 ima tri največje klike* $\{1, 2, 4, 7\}$, $\{2, 4, 5, 7\}$ *in* $\{1, 3, 4, 6, 7\}$. *Zadnja je hkrati tudi maksimalna klika tega grafa.*

Klika $\{1, 3, 4\}$ ni največja, saj je lahko razširjena z vozlišči iz $P = \{6, 7\}$. Dobimo večje klike: $\{1, 3, 4, 6\}, \{1, 3, 4, 6, 7\}, \{1, 3, 4, 7\}$. Tudi klika $\{1, 3, 7\}$ ni največja, čeprav vemo, da $P = \emptyset$. Kliko lahko razširimo z vozlišči iz množice $S = \{4, 6\}$. Dobimo klike $\{1, 3, 4, 7\}, \{1, 3, 6, 7\}$, ki pa so že štete. Edine največje klike so $\{1, 2, 4, 7\}, \{1, 3, 4, 6, 7\}, \{2, 4, 5, 7\}$.

Lema 2.2.10 *Klika Q je največja, natanko tedaj, ko sta obe množici P in S prazni.*



Slika 2.7: Klika in maksimalna klika grafa

Dokaz: Naj bo Q največja klika. Predpostavimo, da $P \neq \emptyset$. Potem obstaja vozlišče $w \in P$, tako da je $vw \in E$, za vsa vozlišvča $v \in Q$, in da je $Q \cup \{w\}$ klika grafa. Iz $P \subseteq V - Q$ sledi, da $Q \neq Q \cup \{w\}$. To pa je v nasprotju s predpostavko, da je Q največja klika. Torej velja, $P = \emptyset$.

Predpostavimo sedaj, da $S \neq \emptyset$. Potem obstaja vozlišče $w \in S$ tako, da je klika $Q \cup \{w\}$ že šteta, kar je ponovno v protislovje s predpostavko, da je Q največja klika. Zato velja, $S = \emptyset$.

Naj bo Q klika, tako da $P = \emptyset$ in $S = \emptyset$. Predpostavimo, da Q ni največja. Potem obstaja klika D, da velja $Q \subseteq D$ in $Q \neq D$. Naj bo $w \in D - Q$. Ker je D klika, $vw \in E$ za vsa vozlišča $v \in D$ in $v \neq w$. Natančno, $vw \in E$ za vsa vozlišča $v \in Q$ in $w \in P$. To je v nasprotju z predpostavko, da je $P = \emptyset$, zato je klika Q največja.

□

Poglejmo si sedaj nekatere od dobro znanih *zgornjih mej* za velikost maksimalne klike [13].

1. Samo vozlišča, ki so vsebovana v množici P lahko razširijo kliko Q:
   $\mathcal{U}_1(Q,P) = |Q| + |P|$.

2. Vozlišče z maksimalno stopnjo v P omeji velikost razširitve klike Q na maksimalno kliko:
   $\mathcal{U}_2(Q,P) = |Q| + max\{\sigma_P(v) + 1 | v \in P\}$.

3. Samo ena povezana komponenta v P lahko razširi kliko C:
   $\mathcal{U}_3(Q,P) = |Q| + |I^{max}|$, kjer I^{max} označuje množico vozlišč največje povezane komponente v $\mathcal{Z}(G_P)$.

4. k-klika vsebuje k vozlišč s stopnjo vsaj $k-1$:
   $\mathcal{U}_4(Q,P) = |Q| + max\{k | \exists v_i < ... < v_k \in P, \delta_P(v_i) \neq k-1\}$.

5. k-klika ima $\frac{k(k-1)}{2}$ povezav:
   $\mathcal{U}_5(Q,P) = |Q| + max\{k \in \mathbb{N} | \frac{k(k-1)}{2} \leq |E_P|\}$, kjer E_P množica povezav grafa $G_P = (P, E_P)$ induciranega z množico vozlišč P.

6. vsako k-kliko z metodo barvanja vozlišč pobarvam z natanko k barvami:
   $\mathcal{U}_6(Q,P) = |Q| + \chi(G_P)$, kjer $\chi(G_P)$ označuje kromatično število grafa induciranega z množico vozlišč P.

Dokazi za meje in nadaljnje informacije o mejah so predstavljeni v referencah [32] za U_1, [4] za U_5 in [34] za U_6. Ostale meje so enostavne razširitve zgornje meje $\mathcal{U}_1$.

Zgornjo mejo za velikost maksimalne klike Q_{max}, ki jo določimo s pravilnim dodeljevanjem barv vozliščem grafa, bomo uporabili v nadaljevanju za iskanje maksimalne klike na neusmerjenem grafu.

2.3 Največji skupni podgraf

Podobnost med dvema grafoma je sorazmerna z velikostjo njunega največjega skupnega podgrafa. Tega konstruiramo tako, da poiščemo bijektivno preslikavo med množicama vozlišč enega in drugega grafa, ki ohranja povezave med vozlišči. Bijektivna preslikava vozlišča preslika tako, da povezana vozlišča prvega grafa slika v povezana vozlišča drugega grafa.

Problema izomorfizma grafov se lotimo tako, da grafe vektorsko preoblikujemo. Te preslikave imenujemo *invariante* grafov. Z njimi bomo definirali tudi povezanost med iskanjem podobnosti v grafih in maksimalno kliko. Poglejmo natančne definicije pojmov, ki smo jih že omenili v poglavju 1.

Definicija 2.3.1 *Naj bosta G_1 in G_2 neusmerjena grafa. Pravimo da sta G_1 in G_2 izomorfna in zapišemo $G_1 \simeq G_2$, če obstaja bijektivna preslikava med vozlišči $f : V_1 \to V_2$ z množico povezav $(v_1, v_2) \in E_1 \iff (f(v_1), f(v_2)) \in E_2$ za vsa $v_1, v_2 \in V_1$. Takšno preslikavo imenujemo **izomorfizem**.*

Izomorfizem ohranja število vozlišč, povezav, stopnje vozlišč, podgrafe in druge lastnosti grafov.

Definicija 2.3.2 *Naj bo $\sigma : \mathcal{G} \longrightarrow \mathbb{R}^d, d \geq 1$ preslikava iz prostora grafov $\mathcal{G}$ v $\mathbb{R}^d$. Če je $G_1 \simeq G_2 \implies \sigma(G_1) = \sigma(G_2)$, potem preslikavo σ imenujemo **invaianta** grafa.*

Velikost grafa je ena izmed invariant grafa. Zanimali nas bodo največji in maksimalni podgrafi, ki jih bomo določili glede na invariante grafov. Grafa lahko pretvarjamo enega v drugega, ne da bi morali pri tem dodati ali izbrisati kakšno povezavo. Če sta grafa izomorfna le na kateri od podmnožic svojih vozlišč in povezav, sta podgrafa iz teh vozlišč in povezav skupna obema grafoma. Podgrafa z največ vozlišči sta največja skupna podgrafa. Problem primerjave dveh grafov lahko definiramo kot:

Definicija 2.3.3 *Podana sta grafa G_1 in G_2 izbrana iz prostora grafov $\mathcal{G}$. Problem primerjave grafov je poiskati takšno funkcijo $s : \mathcal{G} \times \mathcal{G} \longrightarrow \mathbb{R}$, tako da $s(G_1, G_2)$ zadošča podobnosti (različnosti) med grafom G_1 in grafom G_2.*

Definicija 2.3.4 *Če velja* $G_2 \subseteq G_1$ *in če velja* $\forall v, w \in V(G_2) : vw \in E(G_1) \Longrightarrow vw \in E(G_2)$*, potem je* G_2 ***induciran podgraf*** *grafa* G_1*.*

Definicija 2.3.5 *Podgraf* G_1 *grafa* G *je največji glede na invarianto grafov* $\xi(G_1)$*, če ne obstaja nadgraf* G_2 *grafa* G_1 *v grafu* G*, kjer* $\xi(G_2) > \xi(G_1)$*.*

Definicija 2.3.6 *Glede na invarianto grafov* $\xi(G_1)$ *je podgraf* G_1 *grafa* G *maksimalni podgraf, če ne obstaja podgraf* G_2 *grafa* G *v grafu* G*, kjer* $\xi(G_2) > \xi(G_1)$*.*

V kolikor sta dva podana grafa G_1 in G_2 grafa različnih velikosti, zagotova nista izomorfna. Vendar pa je lahko manjši graf, recimo G_2, še vedno podoben grafu G_1, če je G_2 podgraf grafa G_1. Potrebno je poiskati izomorfizem med podgrafoma grafov G_1 in G_2. V tem primeru govorimo o *izomorfizmu podgrafov*. Dva grafa sta podobna, če sta topološko ekvivalentna ali vsebovana drug v drugem. Poglejmo si sedaj še definicije [5] za podobnosti med podgrafi podanih grafov.

Definicija 2.3.7 *Naj bosta* G_1 *in* G_2 *neusmerjena grafa. Graf* G_{pod} *imenujemo skupni podgraf grafov* G_1 *in* G_2*, če je* G_{pod} *podgraf grafa* G_1 *in* G_2*.* G_{pod} *je* ***maksimalni skupni podgraf****, če ne obstaja skupni podgraf grafov* G_1 *in* G_2*, ki bi vseboval večje število vozlišč.*

Primer 2.3.8 *Naj bosta* $G_S = (V_S, E_S)$ *in* $G_T = (V_T, E_T)$ *grafa kot na sliki 2.8 in 2.9. Podana množica vozlišč in množica povezav dveh grafov,*

$V_s = \{v1, v2, v3, v4\}$,
$E_s = \{(v1, v2), (v1, v3), (v1, v4), (v2, v3), (v2, v4), (v3, v4),$
$(v2, v1), (v3, v1), (v4, v1), (v3, v2), (v4, v2), (v4, v3)\}$,
$V_t = \{w2, w3, w5, w6\}$,
$E_t = \{(w2, w3), (w2, w5), (w2, w6), (w3, w5), (w3, w6), (w5, w6),$
$(w3, w2), (w5, w2), (w6, w2), (w5, w3), (w6, w3), (w6, w5)\}$,

predstavlja največji skupni inducirani podgraf teh dveh grafov. Podgraf V_s *je izomorfen podgrafu* V_t *(slika 2.8).*

Podgraf z vozlišči $(V_s \cup \{v5\}$ *in povezavami* $(E_s \cup \{(v4, v5), (v5, v4)\})$ *je izomorfen podgrafu z vozlišči* $(V_t \cup \{w1\}$ *in povezavami* $E_t \cup \{(w1, w2), (w2, w1)\})$. *Izomorfizem med pografoma grafov* $G_S = (V_S, E_S)$ *in* $G_T = (V_T, E_T)$ *predstavlja največji skupni podgraf obeh grafov (slika 2.9).*

Več v [32].

Slika 2.8: Maksimalni skupni inducirani podgraf

Slika 2.9: Maksimalni skupni podgraf

2.4 Produkt grafov

Produkt dveh grafov je graf, katerega množica vozlišč je kartezični produkt množice vozlišč faktorjev. Pravilo, ki določa povezave v produktnem grafu, pa je mogoče izbrati na več načinov. Obstaja več različnih asociativnih produktov. Pogledali si bomo *tenzorski* produkt grafov, ki mu pravimo tudi *kategorični* ali *direktni* produkt [27, 32].

Vozlišča v **produktnem grafu** so urejeni pari (v, w) z $v \in V_1$ in $w \in V_2$,

povezave pa med vozliščema v, w in v', w' če in samo če $v \neq v', w \neq w'$ in če obstaja povezava med v in v' v grafu G_1 in povezava med w in w' v grafu G_2, ali pa povezave med temi vozlišči ne obstajajo.

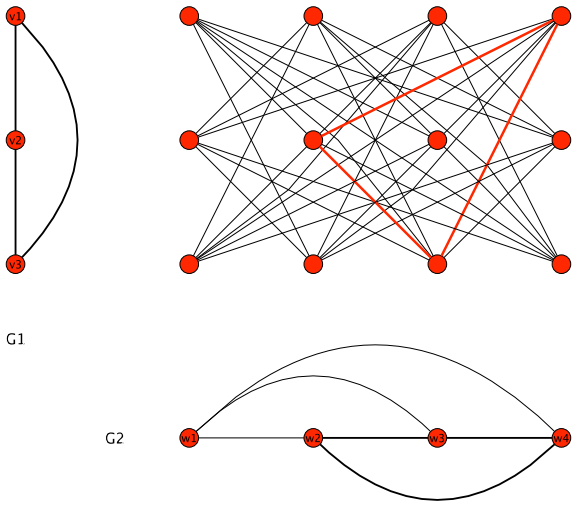
Definicija 2.4.1 *Produktni graf grafov $G_1 = (V_1, E_1)$ in $G_2 = (V_2, E_2)$, ki ga označimo z $G_1 \times G_2$, je graf $G = (V, E)$ z množico vozlišč $V = V_1 \times V_2$ in množico povezav $E = \{((v, w), (v', w')) \in V \times V | v \neq v', w \neq w', (v, v') \in E_1, (w, w') \in E_2\} \cup E = \{((v, w), (v', w')) \in V \times V | v \neq v', w \neq w', (v, v') \notin E_1, (w, w') \notin E_2\}$.*

Direktni produkt $G_1 \times G_2$ grafov $G_1 = (V_1, E_1)$ in $G_2 = (V_2, E_2)$ ima za množico vozlišč kartezični produkt $V(G_1) \times V(G_2)$. Vozlišči (v, w) in (v', w') sta sosednji, natanko tedaj, ko: $v \sim v'$ in $w \sim w'$.

Definicija 2.4.2 *Naj bosta grafa $G_1 = (V_1, E_1)$ in $G_2 = (V_2, E_2)$ in naj bo $Q = (V, E)$ polni podgraf produktnega grafa $G_1 \times G_2$. Preslikava $f : Q \longrightarrow G_1$ je podgraf induciran z vozlišči $v \in V_1$ tako da $(v, w) \in V$ za vozlišča $w \in V_2$. Preslikava $f : Q \longrightarrow G_2$ je podgraf grafa G_2 induciran z vozlišči $w \in V_2$ tako da velja $(v, w) \in V$ za vozlišča $v \in V_1$.*

Primer 2.4.3 *Naj bosta $G_1 = (V_1, E_1)$ in $G_2 = (V_2, E_2)$ grafa kot na Sliki 2.10, z množico vozlišč $V_1 = \{v1, v2, v3\}$ in $V_2 = \{w1, w2, w3, w4\}$ in množico neusmerjenih povezav.*
Maksimalna klika Q_{max} produktnega grafa $G_1 \times G_2$ inducirana na množici vozlišč $\{(v1, w4), (v2, w2), (v3, w3)\}$ ustreza največjemu skupnemu induciranemu podgrafu grafov G_1 in G_2.

Iskanja podobnosti s pomočjo izomorfizmov grafov in podgrafov se poslužujemo pri grafih manjše razsežnosti. Ko število vozlišč in povezav naraste, postane iskanje izomorfizma podgrafov računsko nepraktično, saj narašča tudi velikost podgrafov. Postopek iskanja podobnosti je NP-težek problem in zahteva izčrpajoče preverjanje rešitev ali izomorfizmi res obstajajo.



Slika 2.10: Klika v produktnem grafu

Lahko pa problem iskanja maksimalnega skupnega podgrafa prevedemo na iskanje maksimalne klike v produktnem grafu. Vsaka klika Q v produktnem grafu $G_1 \times G_2$ ustreza skupnemu induciranemu grafu grafov G_1 in G_2. Velikost je enaka pripadajoči kliki Q. Iskanje maksimalne klike Q_{max} v produktnem grafu $G_1 \times G_2$ je ekvivalentno iskanju maksimalnega skupnega podgrafa grafov G_1 in G_2. Rezultat je največja podobnost med dvema strukturama, ki ju primerjamo. Najbolj široko uporaben algoritem za iskanje maksimalnega skupnega podgrafa sta predlagala Bron in Kerbosch [6]. Podrobni predstavitvi algoritma je namenjeno poglavje 4.

Poglavje 3

NP-težki problemi

3.1 Kompleksnost algoritmov in problemov

Kaj je to algoritem? Končno zaporedje ukazov, ki če jih ubogamo, opravijo neko nalogo. Programski jeziki gredo sicer v drugo smer. Razvoj poteka v smeri vse večje abstrakcije. Osnovni način programiranja se glasi: določi procedure, ki jih potrebuješ, in uporabi zanje najboljši možen algoritem. Za rešitev nekega problema potrebujemo torej postopek. Tudi računalnik-CPU razume samo strojni jezik, ki je še najbolj postopkoven, in ne pozna abstrakcije podatkov. Tako brez algoritmov in njihovega poznavanja težko pridemo do rezultatov zahtevnih problemov.

Za algoritem velja:

1. *Ima podatke*
   Vse podatke, s katerimi operira algoritem je potrebno natančno določiti.

2. *Vrne rezultat*
   Vrne vsaj eno izračunano vrednost ali pa opravi neko opravilo.

3. *Je natančno določen*
   Vsak ukaz mora nedvoumno povedati, kaj storiti. Tudi zaporedje izvajanja ukazov mora biti nedvoumno določeno.

4. *Se vedno konča*
   Končati se mora pri vseh možnih naborih vhodnih podatkov.

5. *Mogoče ga je opraviti*
   Načeloma je možno narediti preskus s svinčnikom in papirjem.

V okviru obravnave algoritmov lahko obravnavamo računske stroje, ki opravljajo algoritme, jezike s katerimi opisujemo algoritme, osnove algoritmov, kjer lahko študiramo razrede problemov, kateri problemi so sploh rešljivi, kateri niso časovno zahtevni, lahko pa algoritme tudi analiziramo, in sicer njihovo časovno in prostorsko zahtevnost.

Reševanje problema je odvisno od vrste in velikosti vhodnih podatkov kot tudi od vrste izhodnih podatkov. Izhod algoritma je opis podatkov, ki so v neki meri optimalni glede na vhodne podatke. Očitno je, da algoritem potrebuje dva vira za rešitev problema: *čas* in *prostor*. Čas se običajno meri v številu sprememb stanj v algoritmu od začetka do končanja. Prostor je običajno definiran kot maksimalni prostor potreben za začasne podatke, ki jih algoritem potrebuje preko celotnega računanja. Količina virov, ki jih različni algoritmi potrebujejo za reševanje problema, spada v domeno teorije kompleksnosti. Ta teorija govori o časovni kompleksnosti in o prostorski zahtevnosti.

Kompleksnost algoritma je najdaljši čas potreben za algoritem, da reši problem podane velikosti. To pomeni, da je kompleksnost algoritma merjena za t.i. *wors case*. Torej za najslabši primer, ki je lahko podan. Algoritem s kompleksnostjo največ $\mathcal{O}(x^2)$ bo potreboval štiri krat več časa, če se bo količina podatkov podvojila. Kompleksnost problema je definirana kot kompleksnost najboljšega algoritma za dani problem. Vsi problemi torej niso enako zahtevni. Nekateri problemi so težki in imajo visoko kompleksnost.

Če čas potreben za rešitev problema raste eksponentno z velikostjo primera, je problem težek, saj bo verjetno celo najboljši algoritem neuporaben za realne primere.

3.2 Problem maksimalne klike

V delu opisujem reševanje problema maksimalne klike, ki je dobro poznan problem v teoriji grafov in pravi: za podan graf $G = (V, E)$ poišči kliko grafa z maksimalnim številom vozlišč $\omega(G)$. Problem maksimalne klike sodi v razred NP***-težkih problemov***, za katere ne obstajajo polinomski algoritmi. Za problem eksponentne narave, kot je problem iskanja maksimalne klike, je nujno, da se dokaže njegova NP zahtevnost. Da bi dokazali NP-težkost, bomo najprej definirali pojma *optimizacijski problem* in *odločitveni problem*. Definicija je povzeta iz [1].

Definicija 3.2.1 *Optimizacijski problem P je četverka $P = (I, S, f, cilj)$, kjer*

- *I je množica nalog problema P;*
- *S je funkcija, ki vsaki nalogi $x \in I$ priredi množico $S(x)$ dopustnih rešitev, t.j. prostor možnih rešitev naloge x;*
- *f je kriterijska funkcija, ki vsaki dopustni rešitvi $y \in S(x)$ naloge x priredi vrednost $f(x, y) \in \mathbb{R}$;*
- *cilj pove, ali je treba poiskati dopustno rešitev z najmanjšo (cilj= min) ali največjo (cilj = max) vrednostjo kriterijske funkcije.*

Označimo z y_{opt} optimalno dopustno rešitev naloge x in z $f_{opt} = f(x, y_{opt})$ vrednost kriterijske funkcije, torej optimalno vrednost. Ko govorimo o optimizacijskih problemih, lahko opazimo, da se pojavljajo v treh oblikah. Prva

je *konstrukcijska oblika* problema, ki zahteva, da za dano nalogo x poiščemo optimalno dopustno rešitev y_{opt}. Pri *nekonstrukcijski obliki* optimizacijskega problema zahtevamo, da za dano nalogo x izračunamo vrednost kriterijske funkcije za optimalno rešitev. Tretja oblika, ki je posebej pomembna pri določanju zahtevnosti danega problema, je *odločitvena oblika*, ki za dano nalogo x in konstanto C ugotovi, ali je $f_{opt} \leq C$, če je $cilj = min$ oz. $f_{opt} \geq C$, če je $cilj = max$. Pri odločitvenem problemu ločimo množico dobro definiranih nalog na množico pozitivnih nalog, t.j. takih, za katere je odgovor pritrdilen, in množico negativnih nalog, t.j. nalog, za katere je odgovor negativen.

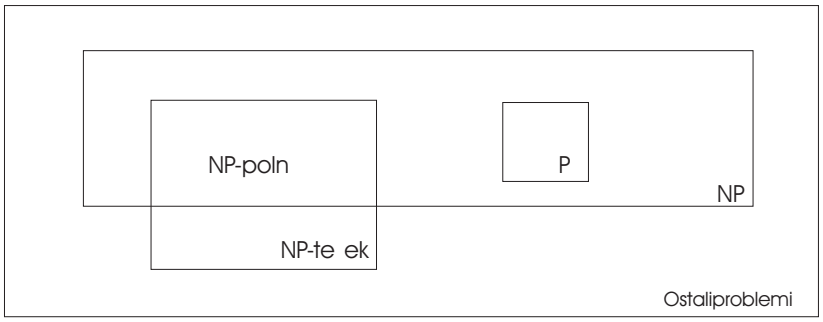
Optimizacijske probleme razdelimo na razreda P in NP, ki ju uvedemo s pomočjo odločitvene oblike danega problema [12]. V razred P (Polynomial time) spadajo odločitveni problemi, ki jih lahko rešimo z algoritmom s kvečjemu polinomsko časovno zahtevnostjo. To so *iskalni* algoritmi, algoritmi *urejanja* (sortiranja) oz. imenujemo jih tudi *poslušni* algoritmi. Razred P lahko vpeljemo s pomočjo poljubnih matematičnih formalizacij za algoritme (npr. Turingov stroj [39]). Izkaže se, da je razred P zelo stabilen glede na različne podrobnosti predpostavk v posameznih modelih. Ti modeli imajo namreč skupno zanimivo lastnost: če lahko problem rešimo v polinomskem času z enim od teh modelov, ga lahko v polinomskem času rešimo tudi v vseh ostalih modelih.

Primeri P problemov:

1. **Povezanost grafa**:
   Vhodni podatki: graf G.
   Vprašanje: Ali je graf G povezan?

2. **Pot v usmerjenem grafu**:
   Vhodni podatki: usmerjen graf $D = (V, E)$, množici $S, T \subseteq V$.
   Vprašanje: Ali obstaja usmerjena pot iz poljubnega vozlišča iz S do poljubnega vozlišča iz T?

Pravimo, da je odločitveni problem v razredu NP (Nondeterministic Polynomial time), če obstaja nedeterministični algoritem, ki ga reši v kvečjemu polinomskem času. Če za odločitveni problem obstaja polinomski algoritem, ki v primeru pozitivnega odgovora preveri pravilnost rešitve, je odločitveni problem v razredu NP. Posebna podmnožica problemov iz NP so ***NP-polni problemi***, za katere velja, da če bi obstajal polinomski algoritem za enega od njih, bi lahko v polinomskem času rešili poljuben problem iz množice NP-polnih problemov. Velja tudi obratno, če bi za enega uspeli dokazati, da ni rešljiv v polinomskem času, bi to veljalo tudi za vse ostale. Vendar še ni dokazano, da polinomski algoritem za NP probleme ne obstaja. Zvezo med problemi razreda P, NP in med NP-polnimi problemi ponazarja slika 3.1.



Slika 3.1: Razredi problemov

Optimizacijski problem NP-težek, če je njegova odločitvena oblika NP-poln problem. Kljub velikemu številu NP-težkih problemov še nihče ni znal za katerega od teh problemov pokazati ali je v razredu P ali ne. Ostaja prepričanje, da je $P \neq NP$ najbolj znan odprti problem teoretičnega računalništva [39].

Definicija 3.2.2 *Problem je NP-poln, če je NP-težek in vsebovan v množici NP.*

Definicija 3.2.3 *Problem je NP-težek, če lahko vsak problem iz razreda NP prevedemo nanj v polinomskem času.*

Poglejmo si sedaj optimizacijski problem maksimalne klike. Vsebuje določitev optimalne vrednosti (maksimum) za podano kriterijsko funkcijo. Sledi še pripadajoči odločitveni problem.

1. **Problem maksimalne klike**
   Vhodni podatki: graf G.
   Vprašanje: Poišči kliko Q z maksimalnim številom vozlišč na grafu G.

2. **Problem klike**
   Vhodni podatki: graf G, parameter $k \in \mathbb{N}$.
   Vprašanje: Ali obstaja klika Q velikosti vsaj k na grafu G?

Da pokažemo, da je problem maksimalne klike NP-težek si pomagamo s prevedbo odločitvenega problema klike na *Booleanov problem izpolnljivosti (3SAT - Boolean satisfiability problem)*. Predvsem v matematični logiki je poznana logika izjav. Izjave so stavki, ki jih delimo po vsebini na resnične (pripada jim logična vrednost 1) in neresnične (pripada jim logična vrednost 0), ter po strukturi na sestavljene in osnovne. Sestavljamo jih z izjavnimi vezniki npr. in, ali, itd. Resničnost sestavljene izjave pa je odvisna od resničnosti njenih sestavnih delov.

Booleanov problem resničnosti izjave je prvi odločitveni problem za katerega je bilo dokazano, da je NP-poln. O tem govori *Cookov izrek* (znan tudi kot *Cook-Levinov izrek*). Dokaz izreka tu opuščamo. Povzetek dokaza je opisan v [39].

Definicija 3.2.4 *Problem A lahko prevedemo na problem B, če za rešitev s problema B obstaja polinomsko izračunljiva funkcija g, tako da je $g(s)$ rešitev problema A. Če lahko rešimo B, lahko rešimo tudi A, kar zapišemo kot $A \longrightarrow B$.*

Definicija 3.2.5 *Če je $A \longrightarrow B$ in $B \longrightarrow A$, potem sta problema B in A ekvivalentna.*

Trditev 3.2.6 *Klika grafa je NP-poln problem.*

Trditev 3.2.7 *Maksimalna klika grafa je NP-težek problem.*

Dokaz obeh trditev 3.2.6 in 3.2.7: Najprej lahko pokažemo, da je problem klike vsebovan v razredu NP. Odločitveni problem je v NP, če je rešljiv z nedeterminističnim algoritmom v polinomskem času. To lahko enostavno vidimo, saj lahko vrednost ciljne funkcije izračunamo v polinomskem času. Preverimo ali naključna množica tvori k-kliko. Množica rešitev je velikosti $\mathcal{O}(k) = \mathcal{O}(n)$. Časovna zahtevnost je $\mathcal{O}(n^2)$.

V drugem delu dokaza moramo pokazati, da bi bil problem maksimalne klike rešljiv v polinomskem času, če bi bil pripadajoči odločitveni problem klike rešljiv v polinomskem času. To lahko naredimo tako, da sestavimo polinomsko prevedbo problema klike na problem, ali je Booleanova izjava resnična [39]. Naj $x_1, x_2, ...$ označujejo množico booleanovih spremenljivk, ter naj $\bar{x}_1$ označuje negacijo x_1. Spremenljivkam lahko zapišemo vrednosti:

1. $x_i \leftarrow 1$, če obstaja vozlišče v U označeno z x_i.
2. $x_i \leftarrow 0$, če obstaja vozlišče v U označeno z $\bar{x}_i$.

Imenujemo jih literali. Izjava H je zgrajena iz stavkov (predikatov), te pa sestavljajo povezani literali. Z uporabo operacij *in* in *ali*. Odločitveni problem $3SAT$ (Boolean satisfiability problem) je problem določiti, ali je izjava H resnična za resnične vrednosti spremenljivk. Z uporabo *Cookovega* izreka sestavimo Booleanovo izjavo H za problem klike. Ta izjava vsebuje n spremenljivk s k stavki (predikati), recimo jim $C_1, ..., C_k$.

$$(x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee \bar{x}_3 \vee \bar{x}_4) \wedge (\bar{x}_1 \vee x_3 \vee x_4) \wedge (x_1 \bar{x}_2 \vee \bar{x}_4)$$

Za izjavo H konstruiramo $k - delni$ graf G_H, kjer so vozlišča literali izjave H. Vsak stavek (predikat) pripada trem vozliščem. Zgradimo graf $G_H = (V_H, E_H)$ v katerem velja:

$$V_H = \{(x_i, C_j) | x_i \in C_j\}$$

$$E_H = \{\{(x_i, C_j, (\bar{x}_i, C_k)\} | C_j \neq C_k \quad \text{in} \quad x_i \neq \neg x_i\}$$

Povežemo vse literale, ki niso v istem stavku in niso negacija drug drugega. Graf G_H lahko prevedemo v polinomskem času na izjavo H. Primer prikazuje slika 3.2. Oznaka $*$ pomeni negacijo literala.

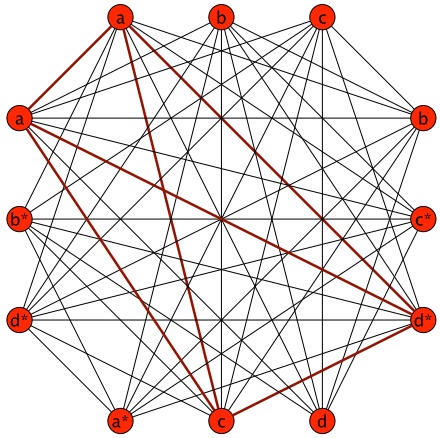
Pokažemo da je izjava H sestavljena iz k-stavkov resnična natanko takrat, ko graf G_H vsebuje kliko Q velikosti k.

$\Longrightarrow$ Predpostavimo, da je izjava resnična. Potem je resničen vsak izmed k stavkov in zato v vsakem stavku obstaja literal, ki je resničen. Množica Q je ravno klika velikosti k na grafu G_H. Množica resničnih literalov, kjer vsak prihaja iz enega stavka, ne more vsebovati nekega literala in njegove negacije in vsi literali so povezani drug z drugim, saj smo povezali literale, ki so v različnih stavkih.

$\Longleftarrow$ Predpostavimo sedaj da je $Q \subseteq V_H$ klika velikosti k v grafu G_H. Ker vemo, da je G_H k-delen graf, mora teh k vozlišč prihajati iz različnih stavkov, ker nobena dva literala iz istega stavka nista povezana. Kot tudi noben literal in njegova negacija nista v kliki. Tedaj so lahko vsi literali v kliki Qresnični in ti nam dajo resnično izjavo H.

□

Pomembna posledica izreka je, da če bi imeli polinomski algoritem za Booleanovo izpolnjivost, potem bi imeli algoritem, ki bi vsak problem v NP rešil v polinomskem času.



Slika 3.2: Problem 3SAT prevedemo na problem klike

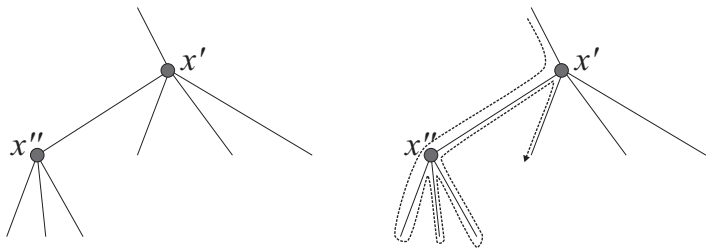
Kadar je najboljša rešitev algoritma nedosegljiva v ustreznem času zaradi velike količine podatkov, moramo opustiti iskanje optimalne rešitve. V praksi večina problemov ne zahteva optimalne rešitve. Zaželena je le dobra rešitev problema. S takim pogledom na problem, je iskanje dobrega aproksimacijskega algoritma povsem dopustno.

3.3 Hevristike

Čeprav je težko definirati hevristiko in bi bila taka definicija precej subjektivna, lahko rečemo, da je to način reševanja problema, ki se izogiba očitnim napakam. Hevristike so strategije, ki vodijo iskalni proces. Cilj je učinkovito preiskovanje iskalnega prostora, v želji po skoraj optimalni rešitvi. Omenimo sedaj nekatere izmed njih.

Sem uvrščamo ***požrešne algoritme***. Ti izberejo posamezno razvrstitev z nekim optimizacijskim pravilom in ne spreminjajo predhodnih razvrstitev. Taki algoritmi ne znajo priti iz slepe ulice. Požrešno metodo barvanja smo si pogledali v začetku poglavja 2.

Pri ***sestopanju*** ali ***vračanju*** (ang. backtracking) rešitev problema iščemo korakoma, sestavljamo podrešitev za podrešitvijo, pri čemer sistematično pregledujemo vse možnosti. Omenjeno lahko lepo ponazorimo z drevesom stanj, katerega splošno sliko vidimo na sliki 3.3 (levo). Recimo, da smo do sedaj pri reševanju problema prišli do podrešitve x', kjer moramo pregledati štiri različne možnosti, odločimo pa se za prvo in pridemo do x'', kjer imamo spet naprimer tri možnosti. Ponovno izberemo eno od teh možnosti, sistematično, da vemo, kaj smo že izbrali in kaj še ne. Seznam možnosti, ki jih še nismo izbrali potrebujemo, ko ugotovimo, da naša izbira ni vodila do rešitve ali pa iščemo naslednjo rešitev, da lahko izberemo naslednjo možnost. Prav temu procesu, da se vračamo na prejšnji nivo (v drevesu stanj) in izberemo naslednjo možnost, rečemo sestopanje. Na tak način lahko tudi hitro zavržemo celotno poddrevo možnih rešitev in s tem zmanjšamo iskalni prostor rešitev, samo iskanje pa je videti kakor sprehod po drevesu stanj, slika 3.3 (desno). Klasični primer takšnega pristopa je iskanje poti v labirintu.



Slika 3.3: Metoda sestopanja

Naslednji pristop je izboljšana metoda sestopanja. Imenujemo jo ***"razveji in omeji"*** (ang. branch and bound). Rešitev je le kandidat za končno rešitev. Namesto slepe izbire med možnostmi (razveji), podamo dodatno oceno "obetavnosti" vozlišč in gremo vedno v smeri najbolj obetavne veje ter tako razvijamo drevo stanj. Po nekaj korakih pridemo do kandidata za rešitev z določeno vrednostjo. Ko pridemo do rešitve, oklestimo "*slabe*" veje (omeji). Ko nimamo več aktivnih možnosti za iskanje, postane trenutni kan-

didat za najboljšo možnost, končna rešitev. “Razveji in omeji” je rekurzivna metoda, ki začne v nekem korenu drevesa, potem pa išče rekurzivno za vse podmožnosti in skuša že v začetnih korakih algoritma odkriti veje, ki zagotovo ne bodo prinesle rešitve.

Izkaže se, da so hevristike lahko zelo različne, kar se tiče učinkovitosti. So metode, ki bi ”po zdravi pameti” morale pripeljati do dobre rešitve in nam bodo služile za lažje reševanje zahtevnega problema maksimalne klike na neusmerjenem grafu G.

Poglavje 4

Algoritmi za iskanje maksimalne klike

4.1 Bron-Kerbosch algoritem

Za NP-težek problem maksimalne klike ne poznamo točnih učinkovitih algoritmov in verjetno tudi ne obstajajo, zato se zadovoljimo z algoritmi, ki nam dajo približne rešitve. Obstajajo aproksimacijski algoritmi, ki ne zagotavljajo globalnega optimuma za vsak podan primer, vendar še vedno najdejo rešitev, ki je dokazljivo "blizu"optimalne.

Sledi predstavitev treh algoritmov za približno iskanje maksimalne klike na neusmerjenem grafu. Razložili bomo, kako lahko pravilno uporabljene hevristične metode pripeljejo do čimbolj optimalne rešitve in kako jih izkoristimo za izboljšanje delovanja algoritma. Navedli bomo tiste razlike, ki vplivajo na izboljšan čas iskanja maksimalne klike in natančnejše rezultate.

Razvoj računalniške tehnologije v šestdesetih oz. sedemdesetih letih prejšnjega stoletja je omogočilo testiranje algoritmov na grafih večjih velikosti. Testiranih je bilo mnogo algoritmov za štetje klik grafa. Verjetno najbolj znan algoritem je *Bron-Kerbosch* algoritem. Razvila sta ga Bron in Kerbosch [6] in je zato tudi dobil takšno ime. Algoritem temelji na metodi sestopanja, kot učinkoviti metodi za iskanje rešitev po rekurzivnem drevesu stanj. Prednost algoritma je, da je izključena možnost generiranja iste klike dvakrat. Predvsem pa je učinkovit na področju iskanja klik v tridimenzionalnih molekulskih strukturah [5].

Algoritem operira s tremi različnimi množicami vozlišč C, P in S tako, da za njih velja:

- Množica C vsebuje množico vozlišč, ki pripadajo trenutni kliki. Skozi postopek, ki ga vrši algoritem, lahko množico razširimo z novim vozliščem ali skrčimo z odvzemom vozlišča;

- Množica P vsebuje vozlišča, ki so možni kandidati za razširitev množice C. Vozlišča so sosednja trenutno izbranim vozliščem v C;

- Množica S vsebuje vsa tista vozlišča, ki so bila prej v množici P. Algoritem jih je v predhodnih korakih uporabil za razširitev množice C, sedaj pa so eksplicitno izključena. Množica S omogoči, da nobeno kliko ne štejemo večkrat.

Pomembno je omeniti, da se vsa vozlišča, ki so sosednja vozliščem v C, nahajajo bodisi v množici P bodisi v množici S. Na i-tem koraku v drevesu stanj množici P in S označimo s P_i in S_i. Definicije, ki se navezujejo na najbolj osnovni *Bron-Kerbosch* algoritem smo pojasnili že v poglavju 2.

Vhodni podatki *Bron-Kerbosch* algoritma [6] (glej postopek na sliki 4.1) so: graf $G = (V, E)$ in množice C, P, S. Postopek iskanja klik na neusmerjenem grafu se začne z vozliščem na začetku iskalnega drevesa, s praznima

Procedure BK(C, P, S)
1. Let P be the set $\{u_1, ..., u_k\}$;
2. **if** $P = \emptyset$ **and** $S = \emptyset$
3. **then** *CLIQUE_FOUND*;
4. **else** $i \longleftarrow 1$ **to** k
5. **do** $P \longleftarrow P - \{u_i\}$;
6. $P \longleftarrow P_i$;
7. $S \longleftarrow S_i$;
8. $N \longleftarrow \{v \in V | \{u_i, v\} \in E\}$;
9. **BK** $(C \cup \{u_i\}, P_i \cap N, S_i \cap N)$;
10. $S \longleftarrow S \cup \{u_i\}$;
11. **od**
12. **fi**

Slika 4.1: Različica Bron-Kerbosch algoritma

množicama C in S. Nova vozlišča, ki jih dodajamo kliki, postopoma izbiramo med vozlišči celotnega grafa G. Definiramo $P = V$. Jedro algoritma vsebuje rekurzivno definirano razširitveno funkcijo (operator), ki ima nalogo generirati vse razširitve množice C z možnimi kandidati iz množice P. Te razširitve množice C so torej večje klike na neusmerjenem grafu.

Na vsakem *nivoju* (*level*) i v drevesu stanj algoritem izvrši naslednje korake:

1. Če $P_i = \emptyset$ in $S_i = \emptyset$ je klika najdena (3.), drugače izberi prvega kandidata u_i in ga odstrani iz P_i (4. in 5.);

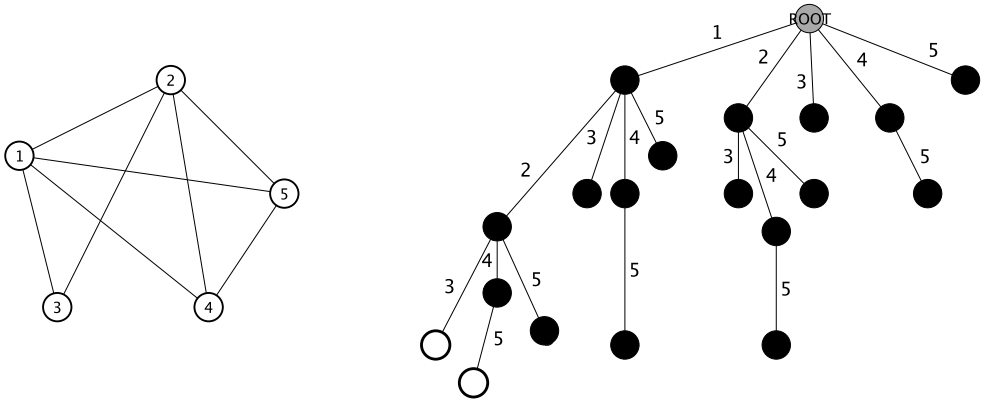
2. Ustvari novi množici P_{i+1}, S_{i+1} iz prejšnjih množic P_i, S_i (6. in 7.);

3. Sosede vozlišča u_i shrani v množico N (8.), ohrani prejšnji množici;

4. Vozlišče u_i doda množici C, razširitvena funkcija (operator), izvede operacijo preštevanja na pravkar ustvarjenih množicah C, P_{i+1} in S_{i+1}.

5. $P_i \cap N$ in $S_i \cap N$ sta nova parametra (9.);

6. Odstranitev vozlišča u_i iz množice C, ki ga doda v staro množico S_i (10.) in nazaj na korak (1.).



Slika 4.2: Primer rekurzivnega drevesa

Slika 4.2 prikazuje iskanje maksimalne klike na grafu s petimi vozlišči z rekurzivnim drevesom, ki ga uporablja *Bron-Kerbosch* algoritem [6]. Povezave v rekurzivnem drevesu (desno) so označene z vozlišči u, ki so dodani trenutni množici C. Sivo obarvano vozlišče označuje začetek algoritma v korenu rekurzivnega drevesa. Pot od korena do vozlišča (bele barve) opiše maksimalno kliko na grafu G. Vsaka pot od korena do kateregakoli razcepnega vozlišča (črne barve) opisuje nek polni podgraf v grafu G, ki ni največji, saj ga lahko razširimo z vsaj enim vozliščem iz neprazne množice S. V tem primeru so klike najdene že v prvih korakih, saj je vozlišče 1, ki je dodano množici C vsebovano v obeh maksimalnih klikah.

Nujni pogoj, da množica C tvori kliko na neusmerjenem grafu je, da je P prazna množica, sicer množico C še vedno lahko razširimo. Vendar ta pogoj ni tudi zadosten, kajti, če množica S še vedno vsebuje kakšno vozlišče, lahko z njim še vedno razširimo množico C. Tako dobimo večjo kliko. Zaradi definicije množice S, da je vsako vozlišče v S sosednje z vsemi vozlišči v C, lahko povzamemo, da je C klika, v primeru, ko sta obe množici P in S prazni. Iz tega definiramo pogoj, ki lahko določi *mejo* za algoritem.

Posledica 4.1.1 *Obstaja vozlišče v množici S, ki je sosednje z vsemi vozlišči v množici P.*

V kolikor na kateri od ravni rekurzivnega drevesa množica S vsebuje vozlišče, npr. v, ki je sosednje z vsemi vozlišči v P, lahko napovemo, da naslednji korak ne bo nikoli pripeljal do boljše rešitve in posledično do izoblikovanja večje klike. Razlog je v tem, da vozlišče v ves čas razširitve od tega koraka dalje ostaja v množici S. Da bi bil pogoj meje izvršen že v začetnih ravneh rekurzivnega drevesa, je potrebno na vsaki razvejitveni točki upoštevati tudi:

1. Določiti vozlišče v iz $S \cup P$ z največjim številom sosedov v P ali najmanjšim številom nesosednjih vozlišč v množici P

2. Najdeno vozlišče $v \in P$ nam predstavlja naslednjega kandidata. Pri sestopanju vozlišče izločimo v množico S.

3. Nadaljujemo z izbiro vozlišč, ki niso sosednja z vozliščem v. Pogoju meje zadostimo, ko se vsa vozlišča, ki niso sosednja vozlišču v nahajajo v množici S. Začnemo s sestopanjem in odstranjevanjem vozlišča.

4.2 Tomita-Seki algoritem

Vsaka nadgradnja in izboljšanje algoritma lahko v veliki meri pripomore k boljši učinkovitosti algoritma. *Tomita-Seki* algoritem [31] uporablja *Bron-Kerbosch* algoritem [6] kot osnovni algoritem in je eden izmed učinkovitih algoritmov za iskanje maksimalne klike na neusmerjenem grafu. Bron-Kerbosch algoritem [6] pa išče vse največje klike na neusmerjenem grafu. Od tega, kako dobra je metoda, ki jo uporablja algoritem, je seveda odvisno, kako dobra je rešitev problema maksimalne klike. Tomita-Seki algoritem [31] upošteva požrešno metodo za približno barvanje vozlišč in primerno zaporedje razvrščanja vozlišč. S tem zagotovi tesno zgornjo mejo za iskanje maksimalne

klike na neusmerjenem grafu. Izpopolnjena metoda približnega barvanja vozlišč lahko torej zelo zmanjša iskalni prostor v rekurzivnem drevesu stanj. To je pomembno predvsem pri grafih, ki vsebujejo veliko število vozlišč in poleg tega spadajo v razred gostih grafov.

Tudi Tomita-Seki algoritem [31] operira z množicami omenjenimi v opisu Bron-Kerbosch algoritma [6]. Podan je graf $G = (V, E)$, v kolikor v nadaljevanju ni posebej omenjeno, predstavlja množica V vozlišča grafa, množici Q in Q_{max} pa množici vozlišč, kjer Q vsebuje vozlišča trenutno nastajajoče klike in Q_{max} vsebuje vozlišča trenutno največje klike. Množica V je urejena, i-ti element v množici pa bomo označevali z $V[i]$. Naj bo $R \subseteq V$ množica, ki vsebuje vsa vozlišča, ki jih je možno dodati v množico Q.

Algoritem začne s prazno množico Q ($Q := \emptyset$) in prazno množico Q_{max} ($Q_{max} := \emptyset$) in nadaljuje z rekurzivnim dodajanjem (odvzemanjem) vozlišč v Q, dokler ne ugotovi, da klike z več vozlišči ni moč poiskati. Vsako naslednje vozlišče, ki je dodano množici Q, je izbrano izmed množice kandidatov vozlišč $R \subseteq V$.

Začetna množica je definirana kot $R := V$. Algoritem izbere vozlišče $p \in R$ in ga doda v množico Q ($Q := Q \cup \{p\}$). Sledi izračun $R_p := R \cap \Gamma(p)$, kot nove množice tistih vozlišč, ki jih algoritem lahko pri naslednjem izboru doda množici Q. V $R \cap \Gamma(p)$ so vsa vozlišča sosednja z vozliščem p. Rekurzivni postopek se nadaljuje vse dokler množica $R_p = \emptyset$, takrat množica Q gradi maksimalno kliko. Če drži $|Q| > |Q_{max}|$, potem se množici Q_{max} in Q zamenjata. V nadaljevanju algoritem z metodo sestopanja odstrani vozlišče p iz Q in iz R. Izbere novo vozlišče p iz ustrezne množice R in nadaljuje postopek vse dokler $R = \emptyset$.

Za izboljšanje osnovnega algoritma 4.3, uporabi algoritem *približno barvanje* vozlišč. S to metodo sta Tomita in Seki [31] določila tesno zgornjo mejo, ki pripomore k odstranitvi vseh tistih vej pri iskanju maksimalne klike Q, ki ne dajo uporabne rešitve že na prvem nivoju rekurzivnega drevesa. Za vsako

Procedure MCQ($G = (V, E)$)
1. **begin**
2. *global* $Q := \emptyset$;
3. *global* $Q_{max} := \emptyset$;
4. Sort vertices of V in a descending order with respect to their degrees;
5. **for** $i := 1$ **to** $\Delta(G)$
6. **do** $N[V[i]] := i$ **od**
7. **for** $i := \Delta(G) + 1$ **to** $|V|$
8. **do** $N[V[i]] := \Delta(G) + 1$ **od**
9. EXPAND (V,N);
10. **output** Q_{max}
11. **end** $\{ofMCQ\}$

Slika 4.3: Osnovni MCQ algoritem

vozlišče $p \in R$ se vnaprej določi pozitivno število $N[p]$, ki ga imenujemo *barva (Numbers)* ali *število vozlišča* p in ima naslednje lastnosti:

i) če $(p, r) \in E$, potem $N[p] \neq N[r]$ in

ii) $N[p] = 1$ ali $N[p] = k > 1$, obstajajo vozlišča

$$p_1 \in \Gamma(p), p_2 \in \Gamma(p), ..., p_{k-1} \in \Gamma(p)$$

v R z barvami

$$N[p_1] = 1, N[p_2] = 2, ..., N[p_{k-1}] = k - 1.$$

Vemo, da velja $\omega(R) \leq Max\{N[p]|p \in R\}$. V $Max\{N[p]|p \in R\}$ so vse *barve* vozlišč in predstavlja t.i. kromatično število grafa. Torej v primeru, če velja

$$|Q| + Max\{N[p]|p \in R\} \leq |Q_{max}|,$$

takšne množice R ne upoštevamo. Vrednost $N[p]$ za vsak $p \in R$ je enostavno dodeljena s pomočjo požrešne metode.

Množico R predstavimo algoritmu kot $R = \{p_1, p_2, ..., p_m\}$. Naj bo $N[p_1] = 1$ in $N[p_2] = 2$, če je $p_2 \in \Gamma(p_1)$, sicer $N[p_1] = 1$, itd. Ko so vsem vozliščem v R dodeljene barve, algoritem razvrsti vozlišča v naraščajočem zaporedju glede na pripadajoče barve. Postopek številčenja in razvrščanja je podrobneje prikazan na Sliki 4.4. Učinkovito dodeljevanje barv je v veliki meri odvisno od tega, kako so vozlišča predstavljena v algoritmu. Zato si poglejmo enostaven postopek za urejanje vozlišč v množico R.
Naj bo

$$Max\{N[r]|r \in R\} = maxno$$

število vseh barv s katerimi obarvamo vozlišča in naj bodo,

$$C_i = \{r \in R|N[r] = i\}, \quad i = 1, 2, ..., maxno$$

vsi barvni razredi vozlišč. Množico R predstavimo algoritmu kot

$$R = C_1 \cup C_2 \cup ... \cup C_{maxno},$$

vozlišča so v R urejena tako kot si sledijo v C_1, nato vozlišča v C_2 in podobno za ostala vozlišča.

Naj bo $C'_i = C_i \cap \Gamma(p), i = 1, 2, ..., maxno$, za $p \in R$ barvni razredi vozlišč, ki so sosednja izbranemu vozlišču iz R in naj velja $R_p = R \cap \Gamma(p) = C'_1 \cup C'_2 \cup ... \cup C'_{maxno}$. Vozlišča v R so urejena kot je opisano zgoraj. Množici C_i in C'_i sta med seboj neodvisni in velja $C'_i \subseteq C_i$, za $i = 1, 2, ..., maxno$. Iz tega je razvidno, da je maksimalna barva potrebna za barvanje množic C'_i manjša ali enaka C_i. Časovna zahtevnost za del algoritma *NUMBER-SORT* je $O(|R|^2)$, kjer zadnji del $\{SORT\}$ razvrsti vozlišča v linearnem casu $O(|R|)$.

```
Procedure NUMBER-SORT (R, N)
 1.   begin
 2.   {NUMBER}
 3.       maxno := 1;
 4.       C_1 := ∅; C_2 := ∅;
 5.       while R := ∅ do
 6.           p := the first vertex in R;
 7.           k := 1;
 8.           while C_k ∩ Γ(p) ≠ ∅
 9.               do k := k + 1 od
10.           if k > maxno then
11.               maxno := k;
12.               C_{maxno+1} := ∅
13.           fi
14.           N[p] := k;
15.           C_k := C_k ∪ {p};
16.           R := R − {p}
17.       od
18.   {SORT}
19.       i := 1;
20.       for k := 1 to maxno do
21.           for j := 1 to |C_k|do
22.               R[i] := C_k[j];
23.               i := i + 1
24.           od
25.       od
26.   end {NUMBER − SORT}
```

Slika 4.4: NUMBER-SORT PROCEDURA

V postopku na sliki 4.5 vidimo, da ponovitev postopka *NUMBER-SORT* zgradi maksimalno kliko z vozliščem $p \in R$, tako da zanj velja $N[p] \geq \omega(R)$. Pričakovati je, da za to vozlišče $p \in R$ za katerega velja

$$N[p] = Max\{N[q]|q \in R\}$$

obstaja velika verjetnost, da pripada maksimalni kliki. Potemtakem, algoritem vedno izbere vozlišče $p \in R$, tako da velja $N[p] = Max\{N[q]|q \in R\}$, kar

sledi iz razcepa **while** v algoritmu *EXPAND*(R, N), glej sliko 4.5. Vozlišče p je zadnji element v urejeni množici R po izvršeni aplikaciji *NUMBER-SORT*. Za to vozlišče velja, da ga algoritem najprej izbere iz množice R. Podobno velja za ostala vozlišča vse dokler $R \notin \emptyset$.

Procedure EXPAND(R, N)
1. **begin**
2. **while**$R \neq \emptyset$ **do**;
3. $p :=$ the vertex in R such that $N[p] = Max\{N[q] | q \in R\}$;
{i.e. the *last* vertex in R
4. **if** $|Q| + N[p] > |Q_{max}|$ **then**
5. $Q := Q \cup \{p\}$
6. $R_p := R \cap \Gamma(p)$
7. **if** $R_p \neq \emptyset$ **then**
8. NUMBER-SORT(R_p, N');
{the initial value of N' has no significance}
9. EXPAND(R_p, N')
10. **else if** $|Q| > |Q_{max}|$ **then** $Q_{max} := Q$ **fi**
11. **fi**
12. $Q := Q - \{p\}$
13. **else return**
14. **fi**
15. $R := R - \{p\}$
16. **od**
17. **end** $\{of EXPAND\}$

Slika 4.5: EXPAND PROCEDURA

Vhodni podatki za algoritem približnega barvanje so vozlišča, ki so v množici V razvrščena v padajočem zaporedju glede na stopnjo vozlišč. Vozlišče z najmanjšo stopnjo bo algoritem izbral najprej. Izbranim vozliščem v množico R pa bo dodelil barve (*Numbers*), tako da velja

$$N[V[i]] = i \quad \text{za} \quad i \leq \Delta(G)$$

in

$$N[V[i]] = \Delta(G) + 1 \quad \text{za} \quad \Delta(G) + 1 \leq i \leq |V|.$$

Začetno dodeljevanje barv ima to lastnost, da pri funkciji *EXPAND(V,N)* velja $N[p] \geq \omega(V)$ za vsak $p \in V$ **while** $V \neq \emptyset$. Glej sliko 4.3. Izračuni, ki so potrebni za učinkovito začetno določitev stopenj vozlišč se izvršijo samo na začetku algoritma predstavljenega na sliki 4.3. S tem se mnogo hitreje zmanjša iskalni prostor, kar omogoči hitrejše iskanje maksimalne klike na neusmerjenem grafu. Hkrati z učinkovitim začetnim razporejanjem vozlišč, ki predstavljajo algoritmu vhodne podatke, pa se ta iskalni prostor še zmanjša.

4.3 MaxClique algoritem

4.3.1 Osnovni MaxClique algoritem

MaxClique algoritem [20], kot so ga poimenovali na Kemijskem inštitutu v Ljubljani, deluje podobno kot že prej omenjeni *Tomita-Seki* algoritem [31]. Razlika je v tem, da so za pohitritev iskanja maksimalne klike na neusmerjenem grafu v njem narejene določene pomembne izboljšave. Te so bistvene za izboljšan čas in delovanje algoritma za iskanje maksimalne klike na neusmerjenem grafu.

Osnovni MaxClique algoritem je predstavljen na sliki 4.6. Za oznako barve vozlišč uporabljajo oznako $C(p)$. Definicija množic Q in Q_{max} ostaja enaka kot pri Tomita-Seki algoritmu **??**. Na vsakem koraku osnovni MaxClique algoritem izbere vozlišče $p \in R$ z maksimalno barvo $C(p)$ izmed vozlišč v R in ga izbriše iz R. $C(p)$ v množici R predstavlja zgornjo mejo za velikost maksimalne klike. Če vsota $|Q| + C(p)$ določi, da obstaja v R večja klika kot trenutno največja Q_{max}, potem vozlišče p dodamo množici Q. Sledi izračun nove množice kandidatov $R \cap \Gamma(p)$ s pripadajočim barvanjem C', ki je dodana kot nov parameter osnovnemu *Maxclique* algoritmu. Če $R_p = \emptyset$ in $|Q| > |Q_{max}|$, dobimo v Q večjo kliko kot je trenutno največja Q_{max} in

vozlišča iz množice Q algoritem preslika v množico Q_{max}. S pomočjo metode sestopanja algoritem na koncu odstrani vozlišče p iz Q in nadaljuje postopek z novim vozliščem iz R, dokler ni $R = \emptyset$.

```
Procedure MaxClique(R, C)
 1.    while R ≠ ∅ do
 2.        choose a vertex p with a maximum color C(p) from set R;
 3.        R := R\{p};
 4.        if |Q| + C(p) > |Q_max| then
 5.            Q := Q ∪ {p};
 6.            if R ∩ Γ(p) ≠ ∅ then
 7.                obtain a vertex-coloring C' of G(R ∩ Γ(p));
 8.                MaxClique(R ∩ Γ(p), C');
 9.            else if |Q| > |Q_max| thenQ_max := Q;
10.            Q := Q\{p};
11.        else return
12.    end while
```

Slika 4.6: Osnovni MaxClique algoritem

4.3.2 Približno barvanje vozlišč

Algoritem za približno barvanje [31] zagotavlja barvanje vozlišč v osnovnem *MaxClique* algoritmu [20]. Vsakemu vozlišču v množici kandidatov R je dodeljena barva v zaporedju kot se pojavi v množici. Algoritem vstavi vsako vozlišče $v \in R$ v prvi možni barvni razred C_k, tako da za vozlišče v velja, da ni sosednje nobenemu vozlišču v tem barvnem razredu. Če ima vozlišče v vsaj enega soseda v vsakem barvnem razredu $C_1, C_2, ..., C_k$, potem je vozlišču v dodeljena nova barva C_{k+1}. Ko algoritem vsem vozliščem dodeli barve, množico teh vozlišč preslika nazaj v množico R. Množica R je sedaj urejena tako, da so v njej zbrani barvni razredi C_k v naraščajočem zaporedju (glede na število k). V tem procesu je barva $C(v) = k$ dodeljena vsem vozliščem $v \in R$. Izhodni podatek algoritma sta novi množici R in C. Barve v množici C pripadajo vozliščem v R.

Zgornja meja za velikost maksimalne klike na neusmerjenem grafu G induciranem z množico vozlišč R je torej število barvnih razredov. To število je odvisno predvsem od načina, kako so vozlišča predstavljena v algoritmu. V kolikor je vhodni podatek za algoritem množica R s padajočim zaporedjem stopenj vozlišč, bo zgornja meja za določitev barvnih razredov tesnejša (lower)

4.3.3 Izboljšan algoritem za približno barvanje grafa

Prva izboljšava osnovnega $MaxClique$ algoritma za iskanje maksimalne klike je torej izboljšana razvrstitev vozlišč v množici R. To pomeni, da bo zaporedje vozlišč v R po dodelitvi barv enako zaporedju vozlišč v R na začetnem koraku algoritma. Kajti prej opisan algoritem, uredi vozlišča v R glede na njihovo barvo, tako da $MaxClique$ algoritem [20] na vsakem koraku izbere iz R vozlišče z maksimalno barvo, kar vedno ustreza zadnjemu vozlišču v množici. Ni pa nujno, da bo to tudi vozlišče z največjo stopnjo.

Očitno je, da tistih vozlišč $v \in R$, ki ustrezajo barvam $C(v) < |Q_{max}| - |Q| + 1$ ni potrebno razvrstiti glede na dodeljeno barvo, ker jih osnovni $MaxClique$ algoritem [20] ne bo dodal nastajajoči kliki Q (Glej korak 4 na sliki 4.6). Namreč ta vozlišča vsebujejo lastnost, da je njihova barva nižje (lower) od barve, ki jo bomo v nadaljevanju označili s k_{min}.

Na začetku algoritma za približno barvanje izračunamo barvo

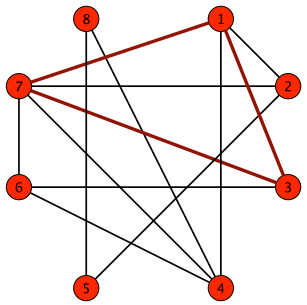
$$k_{min} := |Q_{max}| - |Q| + 1$$

in določimo $j := 0$, kjer je j števec zgoraj omenjenih vozlišč. Če velja $k_{min} \leq 0$, potem določimo $k_{min} := 1$, ker so barve pozitivna števila. V rekurzivnem drevesu stanj je na izhodu vozlišču v na i-tem mestu v R dodeljen barvni razred C_k. Takrat preverimo ali za vozlišče v drži $k < k_{min}$. Če

Procedure ColorSort(R, C)
1. $max\_no := 1$;
2. $k_{min} := |Q_{max}| - |Q| + 1$;
3. **if** $k_{min} \leq 0$ **then** $k_{min} := 1$;
4. $j := 0$;
5. $C_1 := \emptyset$; $C_2 := \emptyset$;
6. **for** $i := 0$ **to** $|R| - 1$ **do**
7. $p := R[i]$; {the i-th vertex in R}
8. $k := 1$;
9. **while** $C_k \cap \Gamma(p) \neq \emptyset$ **do**
10. $k := k + 1$;
11. **if** $k > maxno$ **then**
12. $maxno := k$;
13. $C_{maxno+1} := \emptyset$;
14. **end if**
15. $C_k := C_k \cup \{p\}$;
16. **if** $k < k_{min}$ **then**
17. $R[j] := R[i]$;
18. $j := j + 1$;
19. **end if**
20. **end for**
21. $C[j - 1] := 0$;
22. **for** $k := k_{min}$ **to** max_no **do**
23. **for** $i := 1$ **to** $|C_k|$ **do**
24. $R[j] := C_k[i]$;
25. $C[j] := k$;
26. $j := j + 1$;
27. **end for**
28. **end for**

Slika 4.7: ColorSort algoritem

je prejšnji pogoj izpolnjen, premaknemo vozlišče iz položaja i v položaj j v R, ter povečamo j za 1. Ko je razdelitev v barvne razrede zaključena za vsa vozlišča, so vozlišča z barvami $k < k_{min}$ na začetku množice R. Razvrščena so v padajočem zaporedju glede na stopnjo vozlišč v G. Ostala vozlišča, ki pripadajo barvam $k \geq k_{min}$, algoritem preslika iz barvnih razredov C_k nazaj v množico R, tako kot se pojavijo v vsakem barvnem razredu C_k. V naraščajočem zaporedju glede na število k. Samo tem vozliščem so namreč dodeljene barve $C(v) = k$. Algoritem *ColorSort* je prikazan v postopku na sliki 4.7.



Slika 4.8: Maksimalna klika, $\omega(G) = 3$

Primer dodeljevanja barv

Slika 4.8 predstavlja neusmerjen graf na katerem zelimo poiskati maksimalno kliko. Množica vozlišč v padajočem zaporedju glede na stopnjo vozlišč (v oklepajih) je predstavljena kot

$$R = \{7^{(5)}, 1^{(4)}, 4^{(4)}, 2^{(3)}, 3^{(3)}, 6^{(3)}, 5^{(2)}, 8^{(2)}\}.$$

Ta množica je vhodni podatek za osnovni *MaxClique* algoritem [20]. V tabeli 4.1 je prikazana razdelitev vozlišč v barvne razrede. V vsaki vrstici

so vozlišča barvnega razreda C_k, kjer je število $k \in \mathbb{N}$ barva, ki je dodeljena posameznemu vozlišču.

Ta postopek je enak za oba algoritma, ki uporabljata približno barvanje vozlišč. Po dodelitvi barv, *osnovni algoritem za približno barvanje* preslika vozlišča iz barvnih razredov nazaj v prvotno množico vozlišč R, ki je

$$R = \{7^{(5)}, 5^{(2)}, 1^{(4)}, 6^{(3)}, 8^{(2)}, 4^{(4)}, 2^{(3)}, 3^{(3)}\}$$

s pripadajočim barvanjem

$$C = \{1, 1, 2, 2, 2, 3, 3, 3\}.$$

V primerjavi z množico R iz začetku tega algoritma, je nova množica R manj urejena glede na stopnje vozlišč. V tem algoritmu se popolnoma izgubi urejenost vozlišč glede na stopnjo, saj je za algoritem pomembna razvrstitev vozlišč po naraščajočih barvah. To pa seveda ni dovolj učinkovito za nadaljnji potek algoritma. Neurejenost vozlišč se bo v naslednjih nivojih rekurzivnega drevesa samo še povečevala.

k	C_k		
1	$7^{(5)}$	$5^{(2)}$	
2	$1^{(4)}$	$6^{(3)}$	$8^{(2)}$
3	$7^{(5)}$	$2^{(3)}$	$3^{(3)}$

Tabela 4.1: Barvni razredi

Primer *ColorSort* algoritma

Naj bo $|Q_{max}| := 2$ in $|Q| := 0$. Število $k_{min} = 2 - 0 + 1 = 3$. V delu algoritma, kjer je na vrsti dodelitev barvnih razredov vozliščem, algoritem

premakne vozlišča z barvami $k < 3$, na začetek množice R. Ko so vsa vozlišča razporejena v barvne razrede, dobimo delno množico $R = \{7, 1, 6, 5, 8\}$. Preostanek vozlišč sodi v barvne razrede, kjer je $k \geq 3$. V našem primeru so samo vozlišča iz barvnega razreda C_3 premaknjena v R, v vrstnem redu kot so se pojavila v tem barvnem razredu. Končna množica vozlišč je tako

$$R = \{7^{(5)}, 1^{(4)}, 6^{(3)}, 5^{(2)}, 8^{(2)}, 4^{(4)}, 2^{(3)}, 3^{(3)}\}$$

in pripadajoče barvanje

$$C = \{-, -, -, -, -, 3, 3, 3\},$$

kjer $-$ označuje, da vozlišču ni bila dodeljena barva v množici R. Vozlišča v množici

$$R = \{7^{(5)}, 1^{(4)}, 6^{(3)}, 5^{(2)}, 8^{(2)}\}$$

so razvrščena v padajočem vrstnem redu glede na stopnjo vozlišč. V nasprotju z množico R pri *osnovnem MaxClique algoritmu* imajo ista vozlišča $\{7^{(5)}, 5^{(2)}, 1^{(4)}, 6^{(3)}, 8^{(2)}\}$ drugačno zaporedje. Začetno padajoče razvrščanje vozlišč glede na stopnje, je tako za algoritem ustreznejše. Potrebno ga je ohranjati, kajti število vozlišč v množici kandidatov R, ki pripadajo barvam $k < k_{min}$, je v testiranih grafih v povprečju mnogo večje kot število vozlišč za katere je $k > k_{min}$.

4.3.4 Dinamično barvanje

Do sedaj je bilo razvrščanje in izračun stopenj vozlišč izvršeno le na začetku algoritma. Algoritem za približno barvanje je upošteval vozlišča v množici R urejena glede na njihovo stopnjo v grafu G. *Maxclique* algoritem pa na vsakem koraku ponovno izračuna stopnje vozlišč v induciranem grafu $G(R)$ z množico vozlišč R. Množico vozlišč R pa uredi v padajočem zaporedju glede

na stopnjo vozlišč v $G(R)$. Takšna urejenost vozlišč je tudi bolj ugodna za *ColorSort* algoritem. Zgornje meje, ki jih dobimo s tem algoritmom so še bolj natančne v primerjavi s prvotnim približkom. Število korakov, ki jih potrebujemo za iskanje maksimalne klike se tako zmanjša. Časovna zahtevnost za določitev stopnje in sortiranje vozlišč v R je enaka $O(|R|^2)$.

Časovno zahtevnost algoritma pa lahko izboljšamo s sortiranjem stopnje vozlišč v $G(R)$, ko je množica R zadosti velika. Očitno je, da je množica R večja na začetnih nivojih *MaxClique* algoritma [20]. Zato je potrebno podati natančne zgornje meje že na začetku rekurzivnega drevesa, da se s tem algoritem izogne nepotrebni izgubi časa pri iskanju rešitev v vejah, ki niso ugodne za razširitev maksimalne klike. Algoritem tu upošteva metodo "razveji in omeji". Z izračunom računsko zahtevnejših nalog pri korenu rekurzivnega drevesa lahko omejimo iskanje maksimalne klike. Pri velikih množicah R je čas, ki ga algoritem potrebuje za izračun tesnih mej manjši od časa, ki bi ga potreboval za preiskovanje napačnih rešitev, ki se pojavijo z uporabo manj natančnih mej. Nasprotno pa je pri manjših množicah vozlišč R čas, ki ga algoritem nameni za računanje tesnih mej zelo pomemben. kjer so tesne meje manj pomembne čas, ki bi ga algoritem namenil za določanje tesnih mej zelo pomemben.

Naj *nivo (level)* predstavlja število razvejitev (recursive calls) od začetka korena (root) drevesa do trenutnega lista v rekurzivnem drevesu v *MaxClique* algoritmu [20]. Hitrost algoritma za iskanje maksimalne klike lahko povečamo, če število nivojev, do katerih izračunamo stopnje vozlišč in njihovo razvrščanje določimo dinamično.

Časovna zahtevnost algoritma za iskanje maksimalne klike je odvisna tudi od vrste grafov, na katerih želimo poiskati maksimalno kliko. Na primer, v gostih grafih so v splošnem maksimalne klike večje kot v redkih grafih enakih velikosti. Pričakujemo lahko, da bo število nivojev do katerih naj bi bile uporabljene zahtevnejše računske naloge za določitev tesnih zgornjih mej višje pri gostih kot pri redkih grafih. Ravno tako lahko pričakujemo, da bo

za večje grafe to število nivojev višje kot za manjše grafe enake gostote. V splošnem so maksimalne klike večje v večjih grafih.

Procedure MaxCliqueDyn($R, C, level$)
1. $S[level] := S[level] + S[level-1] - S_{old}[level]$;
2. $S_{old}[level] := S[level-1]$;
3. **while** $R \neq \emptyset$ **do**
4. choose a vertex p with maximum C(p) (*last* vertex) from R;
5. $R := R \backslash \{p\}$;
6. **if** $|Q| + C[index\ of\ p\ in\ R] > |Q_{max}|$ **then**
7. $Q := Q \cup \{p\}$;
8. **if** $R \cap \Gamma(p) \neq \emptyset$ **then**
9. **if** $S[level]/ALL\_STEPS < T_{limit}$ **then**
10. calculate the degrees of vertices in $G(R \cap \Gamma(p))$;
11. sort vertices in $R \cap \Gamma(p)$ in a descending order
12. with respect to their degrees;
13. **end if**
14. ColorSort($R \cap \Gamma(p), C'$)
15. $S[level] := S[level] + 1$;
16. $ALL\_STEPS := ALL\_STEPS + 1$;
17. MaxCliqueDyn($R \cap \Gamma(p), C', level + 1$);
18. **else if** $|Q| > |Q_{max}|$ **then** $Q_{max} := Q$;
19. $Q := Q \backslash \{p\}$;
20. **else return**
21. **end while**

Slika 4.9: MaxCliqueDyn algoritem

Poglejmo si sedaj delovanje *MaxCliqueDyn* algoritma, ki je prikazan na sliki 4.9. Predstavimo najprej globalne spremenljivke $S[level]$ in $S_{old}[level]$. Prva definira vsoto vseh korakov, ki jih *MaxCliqueDyn* algoritem izvrši od začetnega vozlišča do trenutnega nivoja, kjer se algoritem nahaja. Druga spremenljivka $S_{old}[level]$ pa definira vsoto vseh korakov do predhodnega nivoja. Uvedemo spremenljivko $T[level]$,ki primerja vse korake do trenutnega nivoja z vsemi koraki narejenimi do mesta, kjer se algoritem nahaja. Izračunamo jo kot $T[level] = S[level]/ALL\_STEPS$ na vsakem koraku, kjer je ALL_STEPS števec vseh korakov. Ta se na vsakem koraku *MaxCliqueDyn* algoritma

poveča za 1. Uvedemo nov parameter T_{limit} s katerim lahko omejimo izračun tesnih mej do določenega nivoja algoritma.

V primeru, ko je $T[level] < T_{limit}$, izračunamo stopenj vozlišč in izvršimo razvrščanje vozlišč. V *ColorSort* algoritmu pa upoštevamo razvrstitev vozlišč v R glede na stopnje v $G(R)$. Ko je $T[level] \geq T_{limit}$, pa dodatnege računanja ne izvedemo.

Na vsakem nivoju *MaxCliqueDyn* algoritem preračuna vsoto korakov vse do nivoja kjer se nahaja. Ta je enaka $S[level] := S[level] + S[level - 1] - S_{old}[level]$. Vsota korakov do prejšnjega nivoja pa enaka $S_{old}[level] := S[level - 1]$. Vsakič, ko se algoritem povzpne po rekurzivnem drevesu na naslednji nivo, se $S[level]$ poveča za 1.

V tabeli 4.2 je prikazan primer štetja korakov. V stolpcih so predstavljeni nivoji rekurzivnega drevesa. Desnousmerjene puščice predstavljajo pot, ki jo prehodi algoritem, sestopanje pa prikazujejo levo usmerjene puščice. Oznake so $S[level]_{S_{old}[level]}$. Število $S[3] = 5$ izračunamo kot $S[3] + S[2] - S_{old}[3] = 2 + 4 - 2 = 4$, in ker se algoritem povzpne v višji nivo (desna puščica) je $S[3] = S[3] + 1 = 5$ ter $S_{old}[3] = S[2] = 4$.

level	1		2		3	...
	$1_{(0)}$	$\longrightarrow$	$2_{(1)}$	$\longrightarrow$	$2_{(2)}$	
				$\swarrow$		
			$2_{(1)}$			
		$\swarrow$				
	$2_{(0)}$	$\longrightarrow$	$4_{(2)}$	$\longrightarrow$	$5_{(4)}$	$\longrightarrow$
			$\vdots$			

Tabela 4.2: Štetje korakov

4.3.5 Parameter T_{limit}

Parameter T_{limit} so na Kemijskem inštitutu določili s poskusi na naključnih grafih z n vozlišči, kjer je verjetnost da obstajajo povezave med vsakim parom vozlišč enaka p. Testirali so deset naključnih grafov z vozlišči $100 < n < 500$ in verjetnostjo povezav $0.2 < p < 0.99$. Ti grafi so vhodni podatek za *MaxCliqueDyn* algoritem. Algoritem za vsak graf poišče maksimalno kliko, vsakič z različnim parametrom T_{limit}, v intervalu $0.0 < T_{limit} < 1.0$. Ko je $T_{limit} = 0.0$, ne izračunajo tesnih mej, medtem ko pri $T_{limit} = 1.0$ algoritem izračuna tesne meje na vsakem koraku. Izračunali so čase za iskanje maksimalne klike za različne vrednosti parametra T_{limit} in za različne vrednosti n in p. Maksimalno kliko na naključnih grafih s $100, 150, 200$ vozlišči in $p = 0.9$ algoritem najhitreje poišče, ko je vrednost T_{limit} blizu 0.05. To velja za goste grafe (naključne in DIMACS grafe [36]). Za redke grafe je optimalna vrednost $T_{limit} = 0.0$. Izbrali so približno mejo za parameter $T_{limit} = 0.025$. Večje vrednosti tega parametra povečajo čas potreben za izračun maksimalne klike na zgoraj omenjenih redkih grafih. Nižje vrednosti parametra T_{limit} pa upočasnijo iskanje maksimalne klike pri zelo gostih grafih. Druge vrednosti parametra T_{limit} so lahko izbrane v intervalih, ki so prikazani v tabeli 4.3 za posamezne vrste testiranih grafov.

$p\backslash n$	100	150	200	300	500
0.2	0-1	0-0.2	0-0.2	0-0.1	0-0.03
0.3	0-0.5	0-0.2	0-0.5	0-0.03	0-0.005
0.4	0-0.2	0-0.05	0-0.035	0-0.015	0-**0.02**-0.1
0.5	0-0.05	0-0.03	0-**0.01**-0.2	0.001-**0.035**-0.1	0.0001-**0.02**-0.2
0.6	0-0.05	0-0.1	0.005-**0.03**-0.07	0.001-**0.02**-0.2	0.005-**0.035**-0.07
0.7	0.01-**0.035**-0.2	0.005-**0.025**-0.1	0.015-**0.02**-0.2	0.005-**0.05**-0.2	--
0.8	0.005-**0.03**-0.1	0.01-**0.03**-0.1	0.005-**0.05**-0.2	0.005-**0.05**-0.2	--
0.9	0.015-**0.05**-0.2	0.02-**0.05**-0.2	0.025-**0.05**-0.2	--	--
0.95	0.01-**0.035**-0.07	0.01-**0.03**-0.1	0.015-**0.05**-0.1	--	--
0.99	0-1	0-1	0-1	0-1	0-1

Tabela 4.3: Intervali za vrednosti parametra T_{limit} za grafe z n vozlišči in verjetnostjo p.

Izbira parametra T_{limit} je odvisna od načina kako so v algoritmu predstavljena vozlišča. Pri vrednosti parametra $T_{limit} = 0.025$ so zahtevni izračuni stopenj in urejanja vozlišč izvršeni na okoli 2.5% korakov, ki jih naredi *MaxCliqueDyn*

algoritem. Natančno število korakov algoritma lahko niha, ker algoritem spremenljivko $T[level]$ računa dinamično skozi postopek iskanja maksimalne klike.

4.3.6 Numerični rezultati

MaxClique algoritem za iskanje maksimalne klike so testirali na naključno generiranih grafih in na standardnih DIMACS primerih [36], ki so posebej primerni za testiranje učinkovitosti algoritmov za različne probleme diskretne optimizacije, kot je iskanje maksimalne klike, barvanje grafov, itd. DIMACS testne grafe [36] so predstavili leta 1993 na NSF (National Science Foundation) centru za znanost in tehnologijo [37, 36]. Testni grafi so sestavljeni iz naključnih grafov z znanimi velikostmi največjih klik. Na voljo je devet različnih razredov grafov, ki skupaj vsebujejo okoli 66 grafov velikosti od 28 do preko 3.300 vozlišč in z do približno 5.506.380 povezavami. Izboljšava obstoječega algoritma za barvanje grafov [31] je na naključnih grafih odkritje maksimalne klike pospešila do 2,1-krat. Pohitritev je največja pri gostih grafih (verjetnost, da med dvema vozliščema obstaja povezava, je 0.95), to je pri grafih z veliko gostoto povezav. Tu je pospešitev tudi najbolj zaželena, saj je izračun maksimalne klike najbolj dolgotrajen ravno pri tej vrsti grafov. Pri grafih z višjo oziroma nižjo gostoto je pohitritev manjša, a ti grafi so tudi mnogo lažje rešljivi (v manj korakih). Kot smo videli, je bil uveden tudi nov način razvrščanja množice vozlišč, in sicer le na začetnih nivojih rekurzivnega drevesa. Predstavljeni način razvrščanja vozlišč je iskanje na naključnih grafih pospešil od 1.3 do 12-krat v primerjavi z izvornim algoritmom; na DIMACS grafih [36] pa se izkaže različno, vendar je v večini primerov hitrejši in problem reši v manjšem številu korakov kot izvorni algoritem. Čas porabljen za iskanje maksimalne klike in število korakov pri *MaxClique* algoritmu, ki uporablja *ColorSort* algoritem, so prikazani v tabeli 4.4. Podatki v tabeli 4.4

Graf		MCQ		MaxClique+CS		MaxCliqueDyn+CS	
n	p	število korakov	CPU čas	število korakov	CPU čas	število korakov	CPU čas
100	0.6	1031	0.003	978	0.00265	943	0.0028
100	0.7	2752	0.00925	2525	0.0082	1940	0.00735
100	0.8	6674	0.03	5460	0.02445	4101	0.0205
100	0.9	9279	0.0746	6655	0.05405	4314	0.0384
100	0.95	764	0.0092	712	0.00815	477	0.0061
150	0.5	2526	0.0076	2423	0.00695	2217	0.00735
150	0.6	7852	0.0278	7330	0.02525	5932	0.0238
150	0.7	30057	0.1315	27087	0.1164	20475	0.09445
150	0.8	218972	1.2913	174905	1.0368	88649	0.606
150	0.9	1570126	17.478	854110	9.863	258853	3.464
150	0.95	78246	1.7241	40081	0.88255	14825	0.40785
200	0.4	2585	0.00795	2494	0.00745	2409	0.0078
200	0.5	9774	0.0329	9355	0.0304	7695	0.03095
200	0.6	40694	0.1626	38060	0.14845	31114	0.12995
200	0.7	275430	1.383	246297	1.22485	148726	0.8599
200	0.8	3555022	25.725	2850467	20.658	1424940	11.06
200	0.9	145553091	2117.78	80498743	1199.39	16404111	279.705
200	0.95	22393742	729.479	10433846	340.077	1508657	60.524
300	0.4	15014	0.0507	14591	0.0479	12368	0.05335
300	0.5	76112	0.30745	73311	0.28725	61960	0.2634
300	0.6	641131	2.983	603540	2.767	387959	2.133
300	0.7	8521492	51.915	7669432	46.385	4548515	28.566
300	0.8	530043775	4580.5	434614352	3735	159498760	1442.01
500	0.3	23300	0.09055	22785	0.0863	19665	0.10895
500	0.4	141095	0.6202	137348	0.58595	123175	0.5817
500	0.5	1422379	7.009	1368896	6.579	963385	5.827
500	0.6	23889293	142.553	22678735	132.084	15075757	91.447
1000	0.2	45171	0.29045	44766	0.2753	41531	0.3858
1000	0.3	456758	2.386	449717	2.238	413895	2.332
1000	0.4	6192174	34.708	6040135	32.914	4332149	33.761
1000	0.5	140261760	886.465	136018698	842.86	100756405	655.6

Tabela 4.4: CPU čas $[s]$ in število korakov za naključne grafe.

iz [20] prikazujejo, da je $MaxCliqueDyn$ algoritem $1.3 - 12$ krat hitrejši od osnovnega MCQ algoritma. Oba algoritma so testirali na naključnih grafih z verjetnostjo povezav p na intervalu $0.7 - 0.95$. Razlika med algoritmoma je pri drugih grafih manjša. V tabli 4.5 so prikazani rezultati $MaxCliqueDyn$ in osnovnega MCQ algoritma na DIMACS grafih [36]. Čas potreben za izračun maksimalne klike z izboljšanim algoritmom potrjuje, da je parameter $T_{limit} = 0.025$ dovolj natančno izbran. Iskanje maksimalne klike na štirih dokaj zapletenih grafih iz družine `brock800` grafov so rešili v času manj kot 4 ure v primerjavi z osnovnim MCQ algoritmom, ki poišče maksimalno kliko samo v dveh grafih in potrebuje za iskanje rešitve znatno več časa. Podobno velja za družino grafov `p_hat700-3.`Le da v tem primeru osnovni MCQ algoritem sploh ni uspel poiskati maksimalne klike v nobenem izmed grafov. V družini `brock` grafov je čas za iskanje maksimalne klike izboljšan za vse grafe razen enega za skoraj $2 - 3$ krat. Algoritem $MaxCliqueDyn$ je maksimalno kliko v `p_hat1000-2` grafu našel 6-krat hitreje, `p_hat300-3` 3-krat, `p_hat500-3` skoraj 8-krat, `san1000` 7-krat, `san200_0.9_1` skoraj 6-krat, `san400_0.7_2` 6-krat, `san400_0.9_1` 31-krat in `sanr200_0.9` skoraj 6-krat hitreje (glej tabelo 4.5 in 4.6 [20]).

Graf	MCQ			MaxCliqueDyn		
ime	ω	število korakov	CPU čas	ω	število korakov	CPU čas
brock200_1	21	450327	2.74	21	229597	1.56
brock200_2	12	4232	0.017	12	3566	0.018
brock200_3	15	17089	0.088	15	13057	0.0735
brock200_4	17	64332	0.313	17	48329	0.242
brock400_1	27	326153861	2915	27	125736892	1175.68
brock400_2	29	115680020	1204.86	29	44010239	521.01
brock400_3	31	279192244	2297	31	109522985	935.23
brock400_4	33	129575982	1181.96	33	53669377	532.66
brock800_1	≥ 23	956318168	fail	23	1445025793	14344
brock800_2	≥ 24	1071802831	fail	24	1304457116	13507
brock800_3	25	1581256139	17050	25	835391899	9263
brock800_4	26	1105720024	13142	26	564323367	7180
c-fat200-1	12	214	0.0005	12	214	0.0005
c-fat200-2	24	239	0.001	24	239	0.0005
c-fat200-5	58	307	0.003	58	307	0.003
c-fat500-10	126	743	0.0355	126	743	0.0345
c-fat500-1	14	517	0.0015	14	517	0.002
c-fat500-2	26	542	0.0025	26	542	0.003
c-fat500-5	64	618	0.0095	64	618	0.0095
hamming6-2	32	62	0.0005	32	62	0.0005
hamming6-4	4	105	0.0005	4	105	0
hamming8-2	128	254	0.018	128	254	0.018
hamming8-4	16	41603	0.333	16	19107	0.1505
hamming10-2	512	1022	1.3	512	2048	6.63
hamming10-4	≥ 40	16274629	fail	≥ 40	1934328	fail
johnson8-2-4	4	46	0	4	46	0
johnson8-4-4	14	255	0.001	14	221	0.001
johnson16-2-4	8	430130	0.4365	8	643573	0.681
johnson32-2-4	≥ 16	15	fail	≥ 16	15	fail
keller4	11	12209	0.0485	11	8991	0.04
keller5	≥ 27	162625	fail	≥ 27	25921	fail
keller6	≥ 50	40912647	fail	≥ 52	319878688	fail
MANN_a9	16	94	0.0005	16	94	0.0005
MANN_a27	126	38252	6.92	126	38252	7.56
MANN_a45	345	2852231	5480	345	2852231	9037

Tabela 4.5: CPU čas $[s]$ in število korakov na DIMACS benchmark grafih.

Graf	MCQ			MaxCliqueDyn		
ime	ω	število korakov	CPU čas	ω	število korakov	CPU čas
p_hat300-1	8	2137	0.006	8	2084	0.0055
p_hat300-2	25	9944	0.0795	25	7611	0.063
p_hat300-3	36	2519267	30.62	36	629972	9.1
p_hat500-1	9	11275	0.043	9	10933	0.041
p_hat500-2	36	515253	6.57	36	189060	2.81
p_hat500-3	50	239705791	5683	50	25599649	739.16
p_hat700-1	11	34229	0.17	11	27931	0.19
p_hat700-2	44	4355991	87.49	44	1071470	25.4
p_hat700-3	≥ 57	486128224	fail	62	292408292	13583
p_hat1000-1	10	202628	1.04	10	170203	0.9855
p_hat1000-2	46	218545180	5189	46	30842192	859.44
p_hat1000-3	≥ 53	798989871	fail	≥ 59	481023263	fail
p_hat1500-1	12	1283326	8.39	12	1138496	7.75
p_hat1500-2	≥ 55	494607640	fail	≥ 60	405328276	fail
p_hat1500-3	≥ 60	239190830	fail	≥ 69	389162339	fail
san200_0.7_1	30	1542	0.019	30	983	0.0125
san200_0.7_2	18	1577	0.011	18	1750	0.014
san200_0.9_1	70	268746	2.86	70	28678	0.4815
san200_0.9_2	60	489203	5.52	60	72041	1.48
san200_0.9_3	44	1037194	17.9	44	327704	6.66
san400_0.5_1	13	4182	0.0525	13	3026	0.0255
san400_0.7_1	40	135677	2.84	40	47332	0.8195
san400_0.7_2	30	71937	1.8	30	9805	0.286
san400_0.7_3	22	411539	5.8	22	366505	3.21
san400_0.9_1	100	41072828	2108	100	697695	66.43
san1000	15	224566	11.37	15	114537	1.56
sanr200_0.7	18	182244	0.924	18	104996	0.586
sanr200_0.9	42	41107366	583.26	42	6394315	102.66
sanr400_0.5	13	299625	1.34	13	248369	1.14
sanr400_0.7	21	89775740	602.9	21	37806745	287.78

Tabela 4.6: CPU čas $[s]$ in število korakov na DIMACS benchmark grafih.

Poglavje 5

Uporaba v naravoslovju

5.1 Vezavna mesta proteinov

V tem poglavju bomo povzeli uporabo metode iskanja maksimalne klike v naravoslovju, v bioinformatiki, še natančneje pri molekulskem modeliranju. Pristopi za iskanje maksimalne klike na grafih se uporabljajo za iskanje podobnosti med molekulami. Molekule si najlažje predstavljamo sestavljene iz atomov in vezi med atomi, te pa lahko predstavimo s pomočjo teorije grafov. Vsak atom predstavlja vozlišče, vezi med atomi pa ustrezne povezave med vozlišči. Tudi proteini (beljakovine) so naravni primeri označenih grafov. Vsak protein lahko opišemo s pomočjo proteinskih grafov, to je označenih grafov. Proteinska struktura, ki je predstavljena v obliki proteinskega grafa, je primerna za analizo z algoritmi s področja teorije grafov [19].

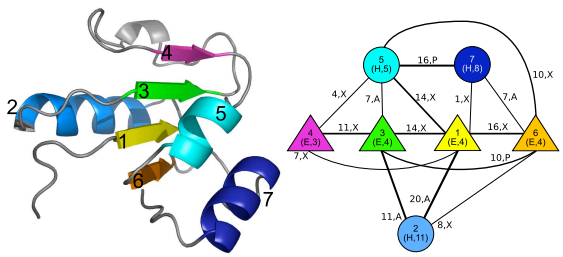
Algoritem za iskanje maksimalne klike je posebej prirejen za primerjave proteinskih struktur, pri čemer je uporabljena lastnost proteinskih grafov, ki se ločujejo od splošnih grafov v tem, da so njihova vozlišča hkrati tudi točke v prostoru. Te točke določijo na osnovi fizikalno-kemijskih lastnosti proteinov.

Primerjava novega algoritma z ostalimi algoritmi za iskanje maksimalne klike v grafu je pokazala, da *MaxClique* algoritem omogoča hitrejše iskanje maksimalne klike [20].

Pri odkrivanju novih zdravilnih učinkovin se farmacija in farmacevtska industrija vedno bolj obračata k molekularnemu modeliranju. Razvoj zdravil je dolgotrajen proces, ki ga poskušajo skrajšati z racionalnim načrtovanjem učinkovin oziroma s spoznavanjem lastnosti bioloških struktur s pomočjo računalnikov. Metode molekularnega modeliranja lahko opredelimo kot metode, ki na osnovi fizikalnega modela napovedujejo biološke procese in pri tem uporabljajo le osnovne biološke, kemijske ali fizikalne podatke o sistemu. V primeru razvoja novih zdravilnih učinkovin s pomočjo računalnikov so zanimive predvsem molekule, vpletene v razvoj različnih bolezeni in njihove interakcije v celici. Cilj je ustvariti takšne modele in simulacije, ki omogočajo napovedovanje lastnosti molekul in njihovih interakcij ter pripomorejo k skrajšanju razvoja novih učinkovin [2, 18, 35].

Interakcije med proteini so odločilnega pomena za delovanje bioloških sistemov. Proteini se lahko povezujejo v komplekse bodisi z majhnimi molekulami, kot so ligandi receptorjev in substrati encimov bodisi z velikimi biološkimi makromolekulami, kot so nukleinske kisline in drugi proteini, pri čemer njihove površinske lastnosti določajo mesto in način vezave. Proteinska vezavna mesta so torej deli strukture na površinah proteinov, kjer se proteini vežejo z drugimi molekulami. Napovedovanje teh mest s pomočjo metod računalniškega modeliranja omogoča vpogled v delovanje in povezovanje proteinov v komplekse. Površina proteinov je torej pomembna, saj prek nje poteka interakcija proteina z okolico. Vezavna mesta za majhne molekule, na primer majhne endogene ligande, se v toku evolucije malo spreminjajo in so pogosto ohranjena v aminokislinskih zaporedjih sorodnih proteinov. Ohranjene so tudi tiste aminokisline, ki so pomembne za stabilizacijo proteinske strukture. Aminokislinski ostanki v vezavnih mestih proteinov imajo točno določeno vlogo pri vezavi in so optimizirani za točno določeno funkcijo vsa-

kega posameznega proteina. Mutacije teh aminokislin se le redko obdržijo, saj napačna aminokislina na takšnem mestu skoraj vedno povzroči porušitev funkcionalno aktivne strukture proteina. Nasprotno pa lahko pričakujemo, da se bo v genskem skladu razširila struktura proteina, ki omogoča prednost tega organizma pred ostalimi. Zaradi velike redkosti takšnih pozitivnih mutacij oziroma genetskih sprememb s pozitivnimi učinki je zelo pogosta ohranjenost za funkcijo pomembnih delov proteinov.



Slika 5.1: Proteinski graf - v tem primeru so vozlišča elementi sekundarne strukture (α vijačnica (krog), β ploskev (trikotnik), povezave pa vezi)

Funkcija številnih na novo kristaliziranih proteinov je nepoznana, a jo je mogoče napovedati s primerjavo s proteini, katerih vloga je že znana. Nekatere metode, s katerimi napovedujemo za funkcijo pomembne predele na proteinski strukturi, temeljijo na dejstvu, da so aminokisline v vezavnih mestih bolj ohranjene kot drugje v proteinu. Ohranjeni deli aminokislinskega zaporedja pri več sorodnih ali celo pri nesorodnih proteinih nakazujejo na določeno vlogo teh aminokislinskih ostankov. Struktura proteinov je še bolj ohranjena kot njihova aminokislinska zaporedja, saj se lahko tudi zelo različna zaporedja zvijejo v med seboj zelo podobne strukture. Za funkcijo proteinov je pomembna predvsem prostorska razporeditev in usmerjenost določenih fizikalno-kemijskih lastnosti, manj pa sama sestava aminokislinskega zapo-

redja. Podobnosti v strukturah so zaradi tega boljši kazalec sorodnosti proteinov, kot so podobnosti v aminokislinskih zaporedjih [15].

Zato s pristopi, ki temeljijo na primerjavah proteinskih struktur, lahko odkrijemo tudi oddaljene sorodnosti, ki jih s primerjavo samih zaporedij ne bi mogli. Uporabnost teh strukturnih metod se še povečuje s hitrim naraščanjem števila določenih 3D struktur proteinov in drugih bioloških molekul v RCSB proteinski bazi [35], v kateri je v tem trenutku že več kot 50,000 bioloških struktur. RCSB proteinska baza vsebuje 3D strukture proteinov in drugih bioloških molekul [35].

5.2 Primerjava proteinskih struktur

Ena izmed metod za določitev vezavnih mest je ta, da se napovedana vezavna mesta za proteine izlušči iz znanih struktur proteinskih kompleksov, in primerja s strukturami proteinov iz RCSB proteinske baze [35]. Strukturo vezavnega mesta primerjamo s površinami ostalih znanih proteinov in vsako najdeno vezavnemu mestu podobno površino vzamemo kot možno novo interakcijsko mesto. Na osnovi podobnosti z znanim vezavnim mestom lahko sklepamo, da mora takšno novo odkrito vezavno mesto tvoriti znanemu kompleksu podoben proteinski kompleks.

Na Kemijskem inštitutu so si za razliko od napovedovanja proteinskih interakcij zastavili drugačen problem, in sicer problem napovedovanja proteinskih vezavnih mest [18, 21, 38]. Za dosego tega cilja pa so morali spremeniti in izboljšati nekatere do sedaj uporabljane pristope. Razvili so algoritem za napovedovanje proteinskih vezavnih mest s pomočjo primerjave proteinskih struktur [21], ki je v obliki spletnega programa dostopen prek spleta. Na spletni strani ProBis [38] so združeni algoritmi za strukturne primerjave proteinov, vključno z *MaxClique* algoritmom [20] predstavljenim v poglavju 4.

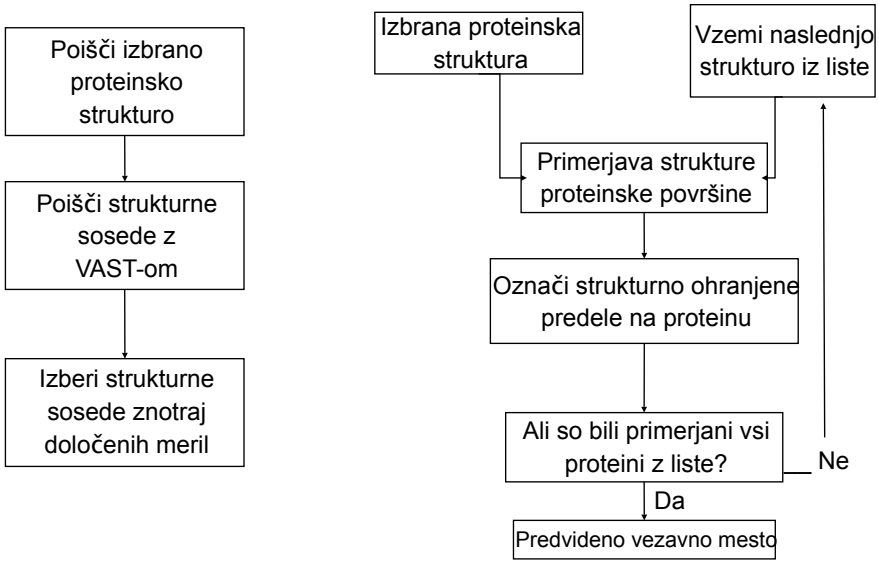
Na začetku izberemo iz RCSB proteinske baze [35] tridimenzionalno strukturo poljubnega proteina, na katerem želimo napovedati vezavna mesta, s katerimi ta protein tvori interakcije z drugimi proteini (slika 5.2 iz [8]). Nato na osnovi predpostavke o evolucijsko povezanih proteinih poiščemo proteine, ki so strukturno sorodni našemu proteinu in od teh izberemo primerne kandidate, s katerimi lahko primerjamo naš protein. Shema algoritma je prikazana na sliki 5.3 in je povzeta po [9].

Slika 5.2: Ujemanje dveh proteinov

Za vsako znano proteinsko strukturo dobimo strukturno podobnega proteina; če strukture še ni v RSCB proteinski bazi [35], lahko poženemo novo iskanje podobnih struktur. Izbrani protein nato z algoritmom za primerjanje struktur primerjamo z vsakim od njegovih strukturnih sosedov, pri čemer na vsakem koraku shranimo najdene ohranjene predele. Vsaki površinski aminokislini izbranega proteina dodamo števec ohranjenosti v strukturnih sosedih.

Algoritem najprej določi topilu dostopne atome (dostopno površino) na obeh proteinskih strukturah [18]. Nato poišče vse aminokisline, ki se nahajajo na površini vsake od obeh primerjanih proteinskih struktur. Na osnovi predpostavke, da se lahko interakcijska mesta pojavljajo kjerkoli na površinah proteinov in da je interakcije proteinov mogoče opisati z enostavnim naborom fizikalno-kemijskih lastnosti, določi ključne fizikalno-kemijske lastnosti funkcionalnih skupin vseh površinskih aminokislin, kot so hidrofobnost, aromatski center in sposobnost tvorbe vodikovih vezi [18]. Površinske aminoki-

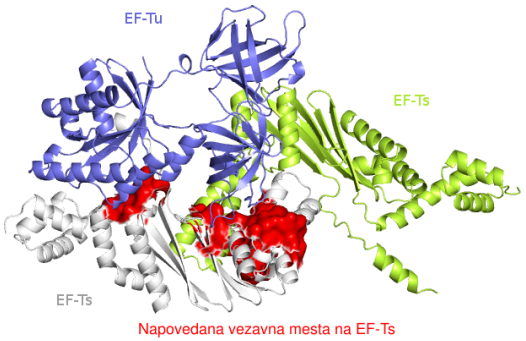
sline so tako opisane z nekaj vozlišči oziroma točkami v prostoru, pri čemer vsaki točki pripada natančno ena fizikalno-kemijsko lastnost. Z vozlišči predstavljeno strukturo proteina nato algoritem pretvori v proteinski graf (slika 5.1 [25]) tako, da tvori povezave med vozlišči, ki so v prostoru oddaljena med sabo za manj od določene razdalje. V nasprotnem primeru povezava ne obstaja. Vsakemu vozlišču priredi matriko razdalj, v kateri so zgoščeno zapisane razdalje do vseh sosednjih vozlišč. To obliko zapisa uporabijo zaradi poenostavitve v nadaljevanju opisane primerjave proteinov. Matrike razdalj skrajšajo čas za dostop do želenih informacij [19].



Slika 5.3: Shema algoritma

V naslednjem koraku algoritem primerja oba proteina, zapisana v obliki proteinskih grafov. Vsako vozlišče prvega proteina primerja z vsakim iz drugega proteina, tako da išče ujemanje njunih oznak. Da zmanjša število ujemajočih se vozlišč, ki je lahko več tisoč, primerjamo namesto vozlišč matrike razdalj

[16].



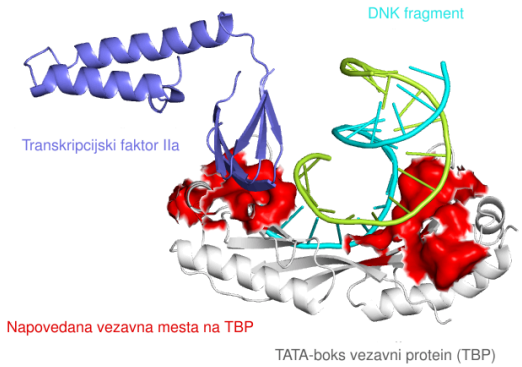
Slika 5.4: Vezavna mesta med proteini

Podobnost med dvema grafoma je sorazmerna z velikostjo njunega največjega skupnega podgrafa. Največji skupni podgraf grafov je mogoče odkriti s prevedbo tega problema na problem iskanja maksimalne klike v tako imenovanem produktnem grafu. Iz dveh proteinskih grafov se tako konstruira produktni graf, pri čemer maksimalna klika v tem grafu predstavlja najbolj podoben del površine dveh obravnavanih proteinov [8].

Algoritem za iskanje maksimalne klike v grafu, ki za vhod potrebuje množico ujemajočih se vozlišč, nato določi največjo ohranjeno podstrukturo obeh površin. Rezultat je seznam v prostoru ujemajočih se vozlišč obeh proteinov, ki ga nato preslikamo v seznam podobnih površinskih aminokislin. Strukturna ohranjenost je najpomembnejši parameter, ki ločuje proteinska vezavna mesta od preostalega dela površine proteinov.

Posebnost novo razvitega pristopa je v tem, da za razliko od obstoječih algoritmov pri iskanju podobnosti med proteinoma ta pristop upošteva celotni površini obeh primerjanih proteinov. Prej zaradi nezadostno razvitega metodološkega koncepta to ni bilo mogoče. Algoritmi zaradi svoje časovne zahtevnosti oziroma kompleksnosti primerjave celotnih površin proteinov sploh

niso mogli obravnavati. Primerjava je bila omejena le na relativno majhne predele proteinskih struktur, to je na vezavne žepe (votline) in na posamezna proteinska vezavna mesta. Pristop napovedovanja vezavnih mest pa omogoči iskanje evolucijsko ohranjenih predelov proteinov tudi na povsem nepredvidenih mestih na površinah proteinov, to je na mestih, ki so nedosegljiva z drugimi pristopi. Na sliki 5.4 so vezavna mesta med strukturno podobnimi proteini in na sliki 5.5 (obe iz [9]) so vezavna mesta med proteini in DNK verigo. Najbolj ohranjeni deli struktur so obarvani rdeče.



Slika 5.5: Vezavna mesta med proteini in dnk molekulo

Zaključek

Sklenimo magistrsko delo s kratkim pregledom čez vsebino. Ukvarjali smo se s problemom maksimalne klike in kako opisati reševanje tega problema s pomočjo nekaterih znanih algoritmov. V začetku magistrskega dela so bili predstavljeni osnovni pojmi iz teorije grafov. Le-ti nam omogočijo definirati kliko na neusmerjenem grafu. Klika na neusmerjenem grafu je poln podgraf induciran na neki izbrani množici vozlišč. Problem maksimalne klike pa je poiskati v grafu kliko z največjim številom vozlišč. Posebej smo obravnavali koncept barvanja grafov, ki smo ga predstavili kot homomorfno preslikavo danega grafa v nek poln graf K_n. Najmanjše število barv s katerimi pobarvamo podan graf predstavlja kromatično število grafa, ki določi najtesnejšo zgornjo mejo za velikost maksimalne klike na neusmerjenem grafu.

V nadaljevanju smo pokazali, da je optimizacijski problem maksimalne klike NP-težek. Problem smo prevedli na znani NP-poln problem *Booleanove resničnostne izjave*. Iz tega rezultata sledi, da je iskanje optimalne rešitve tega problema zelo težka naloga. Omenili smo nekaj najpogostejših hevristik, ki se po javlja jo pri reševanju NP-težkih problemov. Te hevristike so požrešna metoda, sestopanje in metoda "razveji in omeji". Temeljijo na čim boljšem iskanju pravilnih rešitev po rekurzivnem drevesu stanj.

Zaradi praktičnega pomena tovrstnih problemov, kot je problem maksimalne

klike, je bilo do sedaj za reševanje narejenih že veliko algoritmov. Posebno poglavje smo posvetili opisom treh algoritmov. *Bron-Kerbosch* algoritem [6] namenjen iskanju in štetju vseh klik na neusmerjenem grafu. Njegova prednost je, da se izogne večkratnemu štetju iste klike. Algoritem, ki sta ga predstavila Tomita in Seki [31] temelji na približnem barvanju vozlišč. Barvanje vozlišč grafa nam omogoča določiti najtesnejšo zgornjo mejo za velikost maksimalne klike. Algoritem pa z metodo barvanja vozlišč in pravilnim razvrščanjem le-teh, spada med enega izmed najhitrejših algoritmov. Zadnji, na Kemijskem inštitutu razviti $MaxClique$ algoritem [20] pa je izboljšava predhodno omenjenih algoritmov. $MaxClique$ algoritem je bil testiran na naključno generiranih grafih in na standardnih DIMACS grafih [36]. Rezultati nad testnimi grafi so pokazali, da maksimalno kliko poišče v manjšem številu korakov, kar je posledica veliko boljšega razvrščanja vozlišč v primerjavi s *Tomita-Seki* algoritmom [31]. Predstavljen je način razvrščanja vhodnih podatkov algoritma samo na začetnih nivojih rekurzivnega drevesa. Osnovni $MaxClique$ algoritem je nato razširjen, tako da se meje, ki jih algoritem uporablja pri odločanju o nadaljevanju računanja, spreminjajo dinamično. Iskanje maksimalne klike je tako na naključnih grafih pospešeno od 1.3 do 12-krat v primerjavi z izvornim algoritmom.

V zadnjem poglavju smo predstavili uporabo maksimalne klike v naravoslovju za iskanje podobnosti med strukturami. Predvsem tistimi, katerih razsežnost se povzpne tudi nad več kot 1000 vozlišč. S pomočjo teorije grafov smo opisali proteine kot proteinske grafe in predstavili povezavo med proteinskimi grafi in maksimalno kliko. Pokazali smo kako določimo najbolj ohranjen del površine primerjanih proteinov. Funkcionalno najbolj ohranjene dele proteina imenujemo vezavna mesta in ustrezajo velikosti maksimalne klike.

Koncepte teorije grafov smo prikazali kot uporabne za izboljšanje algoritmov v matematiki in bioinformatiki. Grafi pa so se izkazali kot odlično orodje za modeliranje mnogih problemov in se široko uporabljajo tako v matematiki in drugih naravoslovnih vedah.

Literatura

[1] Batagelj, V., *Optimizacijske metode*, Skripta, Ljubljana, 2003, dosegljivo na *http://vlado.fmf.uni-lj.si/vlado/optim/opt1.pdf* (oktober 2008).

[2] Baum, D., *A Point-Based Algorithm for Multiple 3D Surface Alignment of Drug-Sized Molecules*, Dissertation am Fachbereich Mathematik und Informatik der Freien Universit, Berlin, 2007, dosegljivo na *http://www.diss.fu-berlin.de* (oktober 2008).

[3] Baum, D., *Finding All Maximal Cliques of a family of induced Subgraphs*, Technical Report 03-53, Konrad-Zuse-Zentrum, 2003.

[4] Bomze, I. M., Budinich, M., Pardalos, P. M., and Pelillo, M. *The maximum clique problem*, Handbook of Combinatorial Optimization Automatic Acquisition of Domain Knowledge for Information Extraction, In (Supplement Volume A), D.-Z. Du and P. M. Pardalos (Eds.), Kluwer Academic Publishers, Boston, MA, 1999, 1-74.

[5] Borgwardt, K.M., *Graph Kernels*, Dissertation an der Fakultat fur Mathematik, Informatik und Statistik der Ludwig Maximilians Universitat, Munchen, 2007, dosegljivo na *http://edoc.ub.uni-muenchen.de* (oktober 2008).

[6] Bron, C., Kerbosch, J., *Algorithm 457 - Finding All Cliques of an Undirected Graph*, Commun. ACM, 16, 1973, 575-577.

[7] Butenko, S., *Maximum independent set and related problems, with applications*, Dissertation, University of Florida, 2003, dosegljivo na *http://etd.fcla.edu* (oktober 2008).

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

[9] Carl, N., Konc, J., Janežič, D., *Protein surface Conservation in Binding Sites*, Journal of Chemical Information and Modeling, 48, 2008, 1279-1286.

[10] Cazals F., Karande, C., *A note on the problem of reporting maximal cliques*, Theoretical Computer Science, INRIA Tech report 5615, 407, 2008, 1-3.

[11] Corno, F., Prinetto, P., Sonza Reorda, M., *Using Sybolic Tehniques to find the Maximum Clique in Very Large Sparse Graphs*, European Design and Test Conference, 1995.

[12] Diestel, R., *Graph Theory*,Third Edition, Graduate Texts in Mathematics, Springer-Verlag, Heidelberg, 173, 2005.

[13] Fahle, T., *Cost Based Filtering vs. Upper Bounds for Maximum Clique*, Proceedings of the CPAIOR, Paderborn, 2002.

[14] Gross, Jonathan; Jay, Yellen, *Graph Theory and its applications*, CRC Press, Boca Raton, London, New York, Washington D.C., 2006.

[15] Huber W., Carey V. J., Long L., Falcon S., Gentleman R., *Graphs in molecular biology*, BMC Bioinformatics, 8, 2007.

[16] Janežič D., A. Miličevič, A., Nikolič, S., Trinajstič, N., *Graph-Theoretical Matrices in Chemistry*, The University of Kragujevac, Kragujevac, 2007.

[17] Juvan, M., Potočnik, P., *Teorija grafov in kombinatorika*, DMFA, Ljubljana, 2000.

[18] Konc, J., Hodošček, M., Janežič, D., *Molecular Surface walk*, Croat. Chem. Acta, 79, 2006, 237-241.

[19] Konc, J., Janežič, D., *A Branch and Bound Algorithm for Matching Protein Structures*, Lecture Notes in Computer Science, 4432, 2007, 399-406.

[20] Konc, J., Janežič, D., *An Improved Branch and Bound Algorithm for the Maximum clique problem*, MATCH Commun. Math. Comput. Chem., 2007, 58, 569-590.

[21] Konc, J., Janežič, D., *Protein-protein binding-sites prediction by protein surface structure conservation*, Journal of Chemical Information and Modeling 47, 2007, 940-944.

[22] Kosub, S., Local Density, Fakultat fur Informatik, Technische Universitat Munchen, 2004, dosegljivo na *wwwbib.informatik.tu-muenchen.de/infberichte/2004/TUM-I0421.ps.gz* (oktober 2008).

[23] Kutnar, K., Borštnik, U., Marušič, D., Janežič, D., *Interconnection networks for parallel molecular dynamics simulation based on hamiltonian cubic symmetric topology*, J. math. chem., 2008, 1-7.

[24] Makino, K., Uno, T., *New algorithms for Enumerating All Maximal Cliques*, Proc. Ninth Scandinavian Workshop on Algorithm Theory, 2004, 260-272.

[25] May, P., *Protein Structure analysis using contact maps and secondary structure*, Ph. Dissertation, FB Biologie, Chemie, Pharmazie, 2007, dosegljivo na *http://www.diss.fu-berlin.de* (oktober 2008).

[26] Mislej, M., *Homomorfizmi ravninskih grafov z velikim notranjim obsegom*, Diplomsko delo, Fakulteta za matematiko in fiziko, Ljubljana, 2006.

[27] Povalej, Ž., *Produkti grafov*, 2007, dosegljivo na *http://www.fmf.uni-lj.si/∼skreko/dm2/2006-7/Predavanja/Seminarske/ProduktiGrafov.pdf* (oktober 2008).

[28] Rozman, K., *Točkovne grupe in simetrija fulerenov*, diplomsko delo, Pedagoška fakulteta, Ljubljana, 2006.

[29] Strickland, D. M., *Using the Maximum Clique Problem to Motivate Branch-and-Bound*, Informs-Transactions on Educations, 8 (2), 2008, 96-99.

[30] Škrekovski, R., Teorija barvanj grafov,

[31] Tomita, E., Seki, T., *An Efficient Branch-and-Bound Algorithm for Finding a Maximum Clique*, Lecture Notes in Computer science, 2631, 2003, 278-289.

[32] Valiente, G., *Algorithms on Tress and Graphs*, Springer, 2002.

[33] Vishveshwara S., Brinda, K. J., Kannan, N., *Protein structure: Insight from graph theory*, Journal of Theoretical and Computational Chemistry, 1, 2002, 187-211.

[34] Wood, D. R., *An algorithm for finding a maximum clique in a Graph*, Operations Research Letters, 21, 1997, 211-217.

[35] An Information Portal to Biological Macromolecular Structures, *http://www.rcsb.org*, oktober 2008.

[36] Center for Discrete Mathematics and Theoretical Computer Science (DIMACS), *http://dimacs.rutgers.edu/*, oktober 2008.

[37] National Science Foundation - US National Science Foundation (NSF), *http://www.nsf.gov/index.jsp*, oktober 2008.

[38] ProBiS - Protein Binding Sites, *http://tyr.cmm.ki.si/probis/stable/bin/probis.php*, oktober 2008.

[39] Wikipedia, the free encyclopedia, *http://en.wikipedia.org*, oktober 2008.