UNIVERZA NA PRIMORSKEM
FAKULTETA ZA MATEMATIKO, NARAVOSLOVJE IN
INFORMACIJSKE TEHNOLOGIJE

Zaključna naloga

Problemi popolne filogenije

(Perfect Phylogeny Problems)

Ime in priimek: Urša Kačar
Študijski program: Bioinformatika
Mentor: izr. prof. dr. Martin Milanič

Koper, junij 2015

Ključna dokumentacijska informacija

Ime in PRIIMEK: Urša KAČAR

Naslov zaključne naloge: Problemi popolne filogenije

Kraj: Koper

Leto: 2015

Število listov: 51 Število slik: 20
Število referenc: 23

Mentor: izr. prof. dr. Martin Milanič

Ključne besede: popolna filogenija, NP-težek problem, problem barvanja grafov, problem mešane popolne filogenije, hevrističen algoritem

Izvleček:
Eden osnovnih problemov, s katerimi se sooča biologija, je konstrukcija popolnih filogenetskih dreves, ki predstavljajo evolucijsko zgodovino glede na vzorec razvejanja za določene množice objektov. V zaključni nalogi je podrobneje obravnavan problem popolne filogenije, ki se ukvarja z vprašanjem, ali za določene vhodne podatke obstaja popolno filogenetsko drevo, in če obstaja, kako ga konstruiramo. Problem je opisan tako v splošnem kot za posebne primere vhodnih podatkov, npr. na binarnih matrikah. Predstavljena je njegova zveza s problemom barvanja grafov in tetivnimi grafi. Nadalje je obravnavana razširitev problema, ki sta jo leta 2014 formulirala Hajirasouliha in Raphael in jo poimenovala Mešana popolna filogenija. Gre za problem rekonstrukcije zgodovine somatskih mutacij znotraj rakavih celic, pri čemer so vhodni podatki podani z binarno matriko. Opisan je Hajirasoulihov in Raphaelov algoritem, ki temelji na barvanju grafov in ki naj bi rešil zastavljeni problem mešane popolne filogenije, a se je izkazal za napačnega, kot je v nalogi podrobno opisano s protiprimerom. Na koncu je formuliran še nov hevrističen algoritem za rešitev tega problema.

Key words documentation

Name and SURNAME: Urša KAČAR

Title of final project paper: Perfect Phylogeny Problems

Place: Koper

Year: 2015

Number of pages: 51 Number of figures: 20
Number of references: 23

Mentor: Assoc. Prof. Martin Milanič, PhD

Keywords: perfect phylogeny, NP-hard problem, graph coloring problem, Perfect Phylogeny Mixture problem, heuristic algorithm

Abstract:
One of the basic biological problems deals with the construction of perfect phylogeny trees representing evolutionary branching of a certain set of objects. This final project paper discusses the perfect phylogeny problem, the main question of which is whether a perfect phylogenetic tree exists for given input data, and if it does, how can it be constructed. The problem is described both in general and for the special cases of input data, including binary matrices. Its connection with the graph coloring problem and chordal graphs is discussed. Furthermore, a so-called Perfect Phylogeny Mixture problem, formulated by Hajirasouliha and Raphael (WABI 2014) is presented. This problem arises in reconstructing the history of somatic mutations within the cancer cells, and the input is given by a binary matrix. The algorithm by Hajirasouliha and Raphael which is based on graph coloring and which supposedly solves the Perfect Phylogeny Mixture problem is described. However, in this final project paper we show that the algorithm by Hajirasouliha and Raphael is incorrect, as shown with a detailed counterexample. Finally, we propose a new heuristic algorithm for the Perfect Phylogeny Mixture Problem.

Zahvala

Iskrena hvala mentorju, izr. prof. dr. Martinu Milaniču za izjemno strokovno usmerjanje pri izdelavi zaključne naloge, podrobne razlage, potrpežljivost in hitro odzivnost.

Profesorjem in asistentom Fakultete za matematiko, naravoslovje in informacijske tehnologije Univerze na Primorskem se zahvaljujem za neprecenljivo znanje, ki so mi ga posredovali med študijem in mi je bilo pri pisanju zaključne naloge v veliko pomoč.

Na koncu pa bi se rada posebej zahvalila še moji družini, ki me je podpirala skozi vsa študijska leta in mi vedno stala ob strani.

Kazalo vsebine

Kazalo slik

1 Uvod

Osrednja tema zaključne naloge je popolna filogenija, katere glavni problem je določiti, ali je za dane vhodne podatke, predstavljene z matriko, moč konstruirati popolno filogenetsko drevo. Ta drevesa nam za obravnavano množico objektov, kot so recimo populacije, biološke sekvence in drugi taksoni, predstavljajo evolucijsko zgodovino glede na vzorec razvejanja. Čeprav v naravi redko naletimo na biološke podatke, ki ustrezajo vsem pogojem popolne filogenije, je njena teoretična osnova v praksi uporabna za rekonstrukcijo filogenetskih dreves iz nepopolnih, oziroma manjkajočih podatkov, ali pa za modeliranje mutacij znotraj rakavih tumorjev.

Zaključna naloga je razdeljena na tri večja poglavja. Na začetku so podane osnovne definicije grafov, potrebne za razumevanje v nalogi uporabljene terminologije. Posebna pozornost je namenjena dvema tematikama, ki sta za problem popolne filogenije ključnega pomena: barvanju grafov in tetivnim grafom. Za slednje je, poleg njihovih najpomembnejših lastnosti, opisana tudi zanimiva zveza s presečnimi grafi družin poddreves v drevesu, ki sta jo leta 1974 neodvisno predstavila ter dokazala Buneman [6] in Gavril [8], leta 1978 pa še Walter [20].

V poglavju 3 je problem popolne filogenije ustrezno formuliran, nato pa so v prvem podpoglavju predstavljeni njegova povezanost s tetivnimi grafi in problemom barvanja grafov, NP-polnost ter posebna primera vhodnih podatkov, za katera je problem rešljiv v polinomskem času. Drugo podpoglavje se ukvarja s problem karakterizacije instanc popolne filogenije z dvema znakoma. Obravnavan je Gusfieldov polinomski algoritem [10], razvit leta 1991, ki reši vprašanje, ali za določene vhodne podatke, predstavljene z binarno matriko, popolno filogenetsko drevo obstaja, in v primeru pozitivnega odgovora tako drevo tudi konstruira.

Prva polovica poglavja 4 je večinoma povzeta po članku Hajirasoulihe in Raphaela [11], ki sta leta 2014 uvedla razširitev problema, imenovano mešana popolna filogenija. Ta se ukvarja z rekonstrukcijo zgodovine somatskih mutacij znotraj rakavih celic tumorja. Tudi pri tem problemu so vhodni podatki podani v obliki binarne matrike. Razvila sta tudi algoritem za rešitev zastavljenega problema, ki temelji na barvanju grafov. Opisana sta tako algoritem kot primer njegovega delovanja.

Prvotni namen zaključne naloge je bila obravnava te razširitve problema ter praktično testiranje predlaganega algoritma z implementacijo na osnovi hevristik za

barvanje grafov, kot sta npr. DSATUR [5] in Tabucol [12]. Žal se je med pisanjem naloge izkazalo, da je Hajirasoulihov in Raphaelov algoritem napačen in njegova implementacija je bila zato opuščena. Glavni prispevek naloge je tako postal dokaz nepravilnosti algoritma, ki je predstavljen in podrobno opisan s protiprimerom, ter razvoj hevrističnega algoritma za problem mešane popolne filogenije v drugi polovici poglavja 4.

V sodelovanju s soavtorji je avtorica te naloge napisala članek [13], v katerem je problem mešane popolne filogenije podrobneje obravnavan.

2 Osnove o grafih in digrafih

Graf je urejen par $G = (V, E)$, pri čemer je V neprazna množica *točk*, oziroma *vozlišč*, E pa množica *povezav*, pri čemer je povezava množica dveh vozlišč. Kadar za par vozlišč velja $\{u, v\} \in E$, rečemo, da sta vozlišči u in v *sosednji*, oziroma *povezani*.

Graf G je *končen*, če sta množici V in E končni. Za *neusmerjeni graf* G na množici vozlišč $V = V(G)$ in z množico povezav $E = E(G)$ velja, da so povezave neurejeni pari vozlišč, torej $\{u, v\} = \{v, u\}$ za vse $\{u, v\} \in E(G)$. Kadar imajo grafi urejene pare vozlišč, govorimo o *usmerjenih grafih*, ki jim pravimo tudi *digrafi*.

Enostaven graf G je graf, ki ne vsebuje vzporednih povezav (več povezav med parom vozlišč) in zank (povezava na množici dveh enakih vozlišč). Grafi, ki lahko vsebujejo zanke in vzporedne povezave, se imenujejo *multigrafi*. *Polni grafi* so grafi, v katerih je vsako vozlišče povezano z vsakim drugim, *prazni grafi* pa so grafi brez povezav. Kadar splošno govorimo o grafih, imamo v mislih končne enostavne neusmerjene grafe.

Podgraf grafa G je graf H, za katerega velja $V(H) \subseteq V(G)$ in $E(H) \subseteq E(G)$. Rečemo tudi, da je graf G *nadgraf* grafa H. Če za podgraf H velja, da je $V(H) = V(G)$, potem je H *vpet* podgraf grafa G. Vpeti grafi torej vsebujejo vsa vozlišča nadgrafa, ne pa nujno vseh povezav nadgrafa.

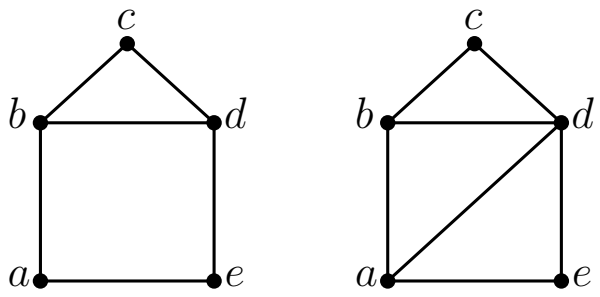
Podgraf H grafa G je *induciran podgraf*, če obstaja taka podmnožica S vozlišč grafa G, da velja $V(H) = S$ in $E(H) = \{\{u, v\} \in E(G) \mid u, v \in S\}$. V tem primeru pravimo, da je H *podgraf grafa* G*, induciran z množico* S, in pišemo $H = G[S]$.

Sprehod v grafu ali digrafu $G = (V, E)$ je zaporedje vozlišč $u_1, u_2, \ldots, u_n$, za katerega velja $\{u_i, u_{i+1}\} \in E$ za vse $i \in \{1, 2, \ldots, n-1\}$. Sprehodu, v katerem so vsa vozlišča različna, rečemo *pot*. *Obhod* je sklenjen sprehod, zanj velja, da je $u_1 = u_n$. Obhodu, v katerem so vsa vozlišča različna, rečemo *cikel*. Graf, ki vsebuje cikel, je *cikličen*, graf brez ciklov pa *acikličen*. Graf G je *povezan*, če med poljubnima vozliščema v grafu obstaja vsaj en sprehod, sicer je *nepovezan*. Povezanemu graf brez ciklov rečemo *drevo*.

V nadaljevanju se pogosto pojavita pojma *klike* in *maksimalne klike*. *Klika* v neusmerjenem grafu $G = (V, E)$ je taka podmnožica točk $C \subseteq V$, da je vsak par točk v C povezan, torej množica paroma povezanih točk grafa. Klika C v grafu G je *maksimalna*, če ni vsebovana v nobeni drugi kliki, torej če iz $C \subseteq C'$ za neko kliko C' sledi $C = C'$. Kliki C, za katero velja, da v grafu G ne obstaja nobena klika, ki bi vsebovala več vozlišč kot C, pa pravimo *največja*.

2.1 Tetivni grafi

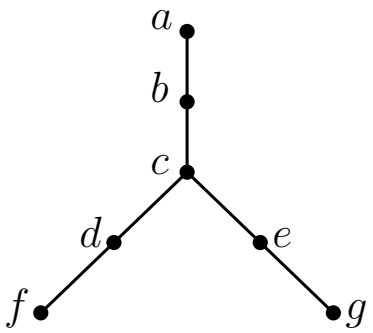
Graf je *tetiven*, če ima v njem vsak cikel dolžine vsaj 4 vsaj eno *tetivo*, tj. povezavo med dvema nezaporednima točkama cikla. Graf na levi strani slike 1 ni tetiven, saj cikel a, b, d, e nima nobene tetive. Z dodatno povezavo med točkama a in d dobimo tetiven graf, prikazan na desni strani. Če je graf G tetiven, so tetivni tudi vsi njegovi inducirani podgrafi.



Slika 1: Netetiven in tetiven graf

Za tetivne grafe se pogosto uporablja tudi izraz *triangulirani grafi*.

Posebna vrsta tetivnih grafov so *intervalni grafi*, definirani kot presečni grafi družin zaprtih intervalov na realni osi. Vsi intervalni grafi so tetivni, medtem ko trditev v drugo smer ne drži. Na sliki 2 je primer tetivnega grafa, ki ni intervalen [9].



Slika 2: Tetiven graf, ki ni intervalen.

Navedimo sedaj nekaj pomembnih lastnosti tetivnih grafov. Buneman [6] in Gavril [8] sta leta 1974 neodvisno drug od drugega dokazala, da so tetivni grafi natanko presečni grafi družin poddreves v drevesih. Isti rezultat je štiri leta kasneje objavil še Walter [20].

Za dano drevo T in poddrevesa $T_1, T_2, \ldots, T_n$ drevesa T definiramo *presečni graf* dane družine poddreves kot graf G z množico točk $\{1, \ldots, n\}$, kjer je $\{i, j\} \in E(G)$ natanko tedaj, ko je $i \neq j$ in imata T_i in T_j vsaj eno skupno točko.

Izrek 2.1 (Buneman [6], Gavril [8])**.** *Če je* $\{T_1, T_2, \ldots, T_n\}$ *družina poddreves drevesa* T*, potem je presečni graf te družine poddreves tetivni.*

Definicija 2.2. Točka v grafu G je *simplicialna*, če je njena soseščina v G klika.

Naslednja pomembna lastnost tetivnih grafov je obstoj *popolne eliminacijske sheme*. Gre za možnost odstranjevanja točk grafa v določenem zaporedju, pri čemer je vsaka odstranjena točka simplicialna glede na reduciran graf, ki ga dobimo z odstranitvijo vseh predhodnih točk. Velja, da ima vsak tetiven graf vsaj eno simplicialno točko in posledično tudi popolno eliminacijsko shemo, saj je vsak induciran podgraf tetivnega grafa tetiven graf. Za dani tetiven graf lahko obstaja več takih shem [9].

Lastnost obstoja popolne eliminacijske sheme je uporabna za prepoznavanje in dokazovanje tetivnosti grafa, saj velja tudi v nasprotni smeri – vsi grafi s to lastnostjo so tetivni. Če torej lahko zaporedoma odstranjujemo simplicialne točke, pri čemer lahko začnemo s poljubno, dokler ne odstranimo vseh, je graf tetiven. Če med odstranjevanjem dobimo graf, v katerem ni nobene simplicialne točke, potem graf ni tetiven [23].

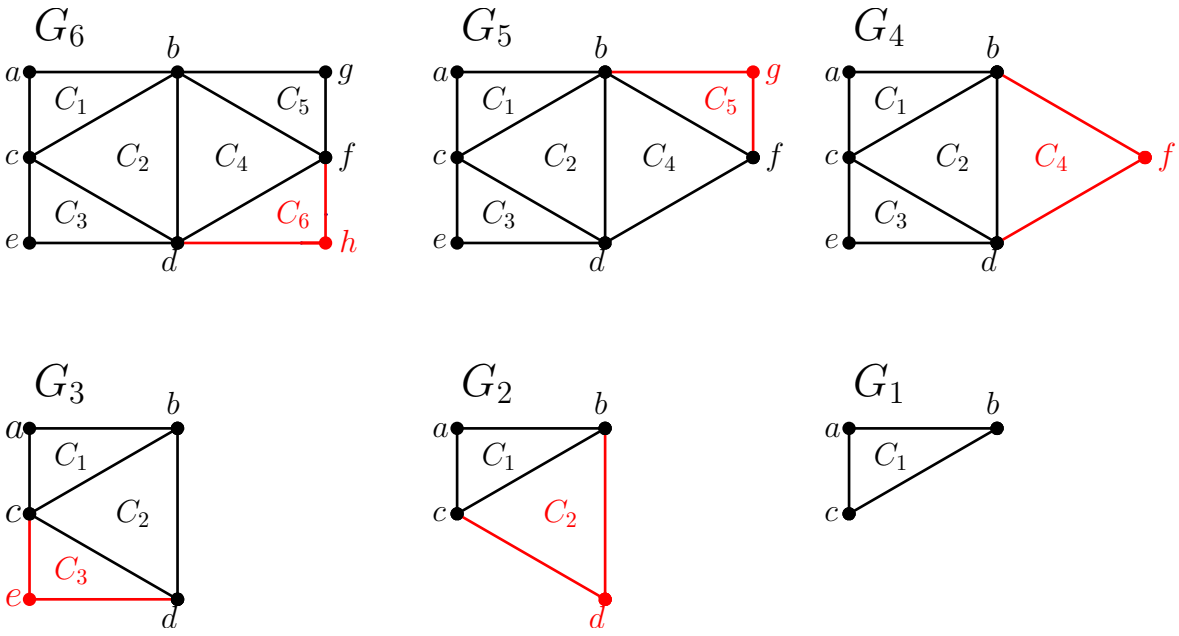
Primer 2.3. Na sliki 3 je prikazan primer odstranjevanja simplicialnih točk za preverjanje tetivnosti grafa. Dan je graf G_6 s šestimi klikami. V prvem koraku odstranimo poljubno simplicialno točko, v našem primeru je to točka h. S tem izgubimo kliko C_6 in kot rezultat dobimo graf G_5. V naslednjem koraku ponovno odstranimo simplicialno točko, v našem primeru g, izgubimo kliko C_5 in dobimo graf G_4. Z nadaljnim zaporednim odstranjevanjem simplicialnih točk f, e in d ter z njimi klik C_4, C_3, C_2, dobimo graf G_1 z eno samo maksimalno kliko, C_1.

Graf G_1 je poln graf. Iz njega lahko v poljubnem vrstnem redu odstranimo še točke a, b in c. Ker nam je z zaporednim odstranjevanjem simplicialnih točk uspelo odstraniti vse točke, smo s tem dokazali, da je graf G_6 tetiven. ▲

Naslednji izrek podaja obrat trditve v izreku 2.1. Če je G tetiven graf, po spodnjem izreku obstaja tako drevo T in družina $\mathcal{T}$ poddreves drevesa T, da je G izomorfen presečnemu grafu družine $\mathcal{T}$.

Izrek 2.4 (Buneman [6], Gavril [8])**.** *Vsak tetiven graf* G *je presečni graf take družine poddreves nekega drevesa* T*, katerega točke ustrezajo maksimalnim klikam grafa* G*, da če točka* c_i *drevesa* T *ustreza kliki* C_i *v* G*, potem poddrevo* T_a*, ki ustreza točki* a *v grafu* G*, vsebuje točko* c_i *če in samo če* C_i *vsebuje* a*.*

Dokaz izreka opustimo, oglejmo pa si primer konstrukcije drevesa T in poddružine dreves za tetiven graf iz zgornjega zgleda (primer 2.3). Dokazali smo že, da je graf G_6 na sliki 3 tetiven, torej mu lahko zgradimo pripadajoče drevo. Začnemo z eno kliko v grafu, ki ji postopoma dodajamo točke, kot je prikazano na sliki 4 in opisano v naslednjem primeru.



Slika 3: Odstranjevanje klik

Primer 2.5. G_1: začnemo z eno samo kliko, C_1, na točkah a, b, c. Pripadajoče drevo T_1 za graf G_1 vsebuje eno samo točko c_1, ki ustreza kliki C_1 v grafu. Poti v drevesu za točke a, b in c, $T_1(a)$, $T_1(b)$, $T_1(c)$, vsebujejo točko c_1, saj so te točke vsebovane v kliki C_1.

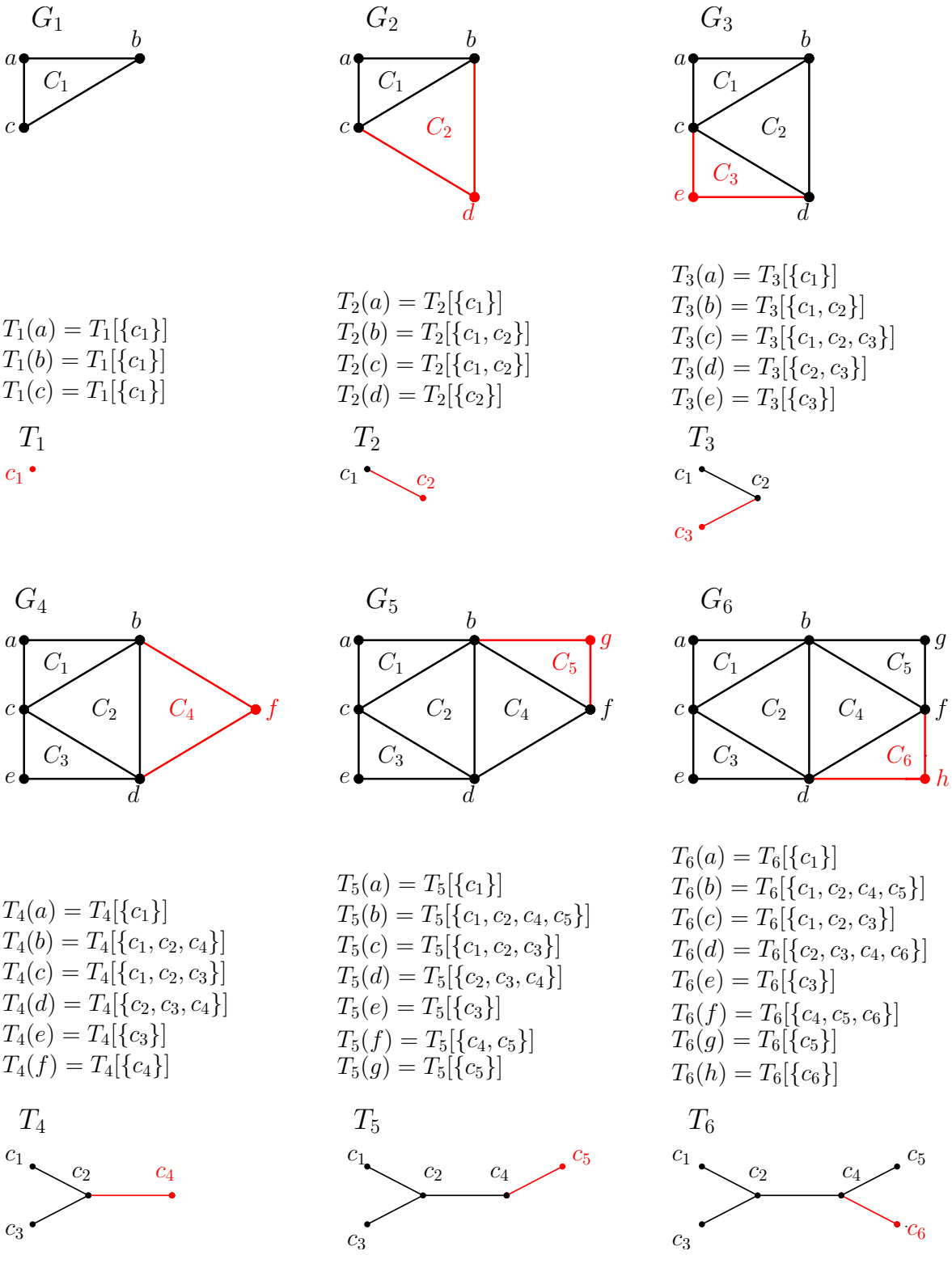
G_2: dodana je točka d, ki ustvari novo kliko C_2 na točkah b, c, d. V drevesu T_2 je dodana točka c_2, ki je povezana s c_1. Potema za točki b in c je dodana točka c_2, saj sta točki v novem drevesu vsebovani v kliki C_2. Dodana je pot točke d, $T_2(d)$, ki vsebuje točko c_2.

G_3: dodana je točka e, ki ustvari novo kliko C_3 na točkah c, d, e. V drevesu T_3 je dodana točka c_3, ki je povezana s c_2. Spremenita se poti v drevesu za točki c in d, dodana je pot točke e, $T_3(e)$, ki vsebuje točko c_3.

G_4: dodana je točka f, ki ustvari novo kliko C_4 na točkah b, d, f. V drevesu T_4 je dodana točka c_4, ki je povezana s c_2. Spremenita se poti v drevesu za točki b in d, dodana je pot točke f, $T_4(f)$, ki vsebuje točko c_4.

G_5: dodana je točka g, ki ustvari novo kliko C_5 na točkah b, f, g. V drevesu T_5 je dodana točka c_5, ki je povezana s c_4. Spremenita se poti v drevesu za točki b in f, dodana je pot točke g, $T_5(g)$, ki vsebuje točko c_5.

G_6: dodana je točka h, ki ustvari novo kliko C_6 na točkah d, f, h. V drevesu T_6 je dodana točka c_6, ki je povezana s c_4. Spremenita se poti v drevesu za točki d in f, dodana je pot točke h, $T_6(h)$, ki vsebuje točko c_6. ▲



Slika 4: Gradnja drevesa

2.2 Barvanja grafov

Barvanje grafa G je definirano kot dodeljevanje barv vozliščem grafa tako, da sta vsaki sosednji vozlišči obarvani z različnima barvama. Problem barvanja grafov se ukvarja z

iskanjem najmanjšega števila barv, s katerim lahko pobarvamo dani graf.

Definicija 2.6. Graf G je *k-obarvljiv*, če obstaja taka preslikava $c : V(G) \to \{1, 2, \ldots, k\}$, da je $c(u) \neq c(v)$ za poljubni sosednji vozlišči $u, v \in V(G)$. Preslikavo c imenujemo *k-barvanje* grafa G. Najmanjše število k, za katero obstaja k-barvanje grafa G, imenujemo *kromatično število* grafa G in ga označimo s $\chi(G)$. Graf je *k-kromatičen*, če velja $\chi(G) = k$.

Problem barvanja grafov je torej ekvivalenten iskanju kromatičnega števila k grafa G. Velja, da je $\chi(G) \leq n(G)$ pri čemer je $n(G)$ število vozlišč grafa G. Zgornja meja kromatičnega števila je torej število vozlišč grafa. Ta zgornja meja je dosežena z enakostjo za polne grafe, v katerih je vsako vozlišče povezano z vsemi ostalimi.

Definicija 2.7. *Klično število* $\omega(G)$ grafa G je največje število vozlišč v kliki grafa G.

Za vsak graf velja, da je $\chi(G) \geq \omega(G)$. Spodnja meja za kromatično število danega grafa je torej klično število grafa. Vozlišča poljubne klike grafa so namreč paroma povezana in morajo biti različnih barv. Enakost $\chi(G) = \omega(G)$ velja za t.i. *popolne grafe*.

Definicija 2.8. Graf G je *popoln*, če za vsak induciran podgraf H grafa G velja $\chi(H) = \omega(H)$.

Vsi tetivni grafi so popolni grafi, zato je njihovo kromatično število enako kličnemu številu. V splošnem pa je problem iskanja kromatičnega števila in s tem optimalnega barvanja grafa NP-težek, oziroma NP-poln, kadar imamo dano število $k > 2$ in nas zanima, ali je dani graf k-obarvljiv [7]. Zaradi velike kompleksnosti problema se za barvanje grafov uporablja hevristične algoritme, kot sta recimo DSATUR [5] in Tabucol [12], zelo znana pa je tudi *požrešna metoda*. Gre za enostaven algoritem, ki postopoma gradi rešitev tako, da v vsaki iteraciji izbere možnost, ki v tistem koraku največ doprinese h končni rešitvi in hkrati še vedno tvori dopustno (delno) rešitev. V primeru iskanja barvanja vozlišč grafa požrešna metoda torej po korakih barva vozlišča v nekem zaporedju tako, da vsako vozlišče obarva z najmanjšo možno barvo. Pri tem pazi, da zadovolji pravilu barvanja grafov, ki zahteva, da sta povezani vozlišči različnih barv. V praksi požrešna metoda daje slabe rezultate, saj je kvaliteta njenih rezultatov odvisna od izbranega vrstnega reda vozlišč. Seveda obstaja neko zaporedje vozlišč, v katerem metoda privede do optimalne rešitve barvanja grafa, a je tako zaporedje težko najti [17]. Izjema so tetivni grafi, ki jim optimalno barvanje najdemo z uporabo požrešne metode na vrstnem redu vozlišč, ki je nasproten vrstnemu redu odstranjevanja vozlišč v neki fiksni popolni eliminacijski shemi [23].

Za graf, opisan v primeru 2.3 in prikazan na sliki 3, bi torej uporabili isti vrstni red, kot smo v primeru 2.5 gradili drevo na sliki 4.

3 Problem popolne filogenije

Eden najosnovnejših problemov biologije je konstrukcija popolnih filogenetskih dreves, ki predstavljajo evolucijsko zgodovino določene množice objektov (to so lahko organizmi, populacije, biološke sekvence in drugi taksoni) glede na vzorec razvejanja, ne nujno glede na čas [2]. Standardno je filogenija predstavljena z množico objektov in množico njihovih lastnosti, ki lahko zavzamejo različna stanja [3].

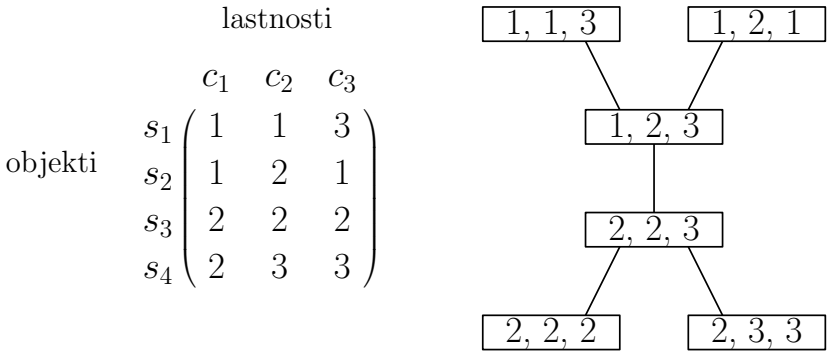
Naj bo $\mathcal{S}$ množica n objektov in $\mathcal{C}$ množica m lastnosti. Posamezni objekt iz $\mathcal{S}$ glede na $\mathcal{C}$ lahko brez škode za splošnost predstavimo z vektorjem $s \in \mathbb{Z}^m$. Stanje lastnosti c za objekt s označimo s $c(s)$, množico vseh dovoljenih stanj pa z $\mathcal{A}$. Tako množico z n objekti in m lastnostmi lahko predstavimo z $n \times m$ matriko M, v kateri so zapisana stanja lastnosti posameznih objektov, pri čemer polje (i, j) predstavlja stanje lastnosti j na objektu i. Primer take matrike je prikazan na sliki 5 levo.

Filogenetsko drevo za dane podatke $I = (\mathcal{S}, \mathcal{C})$ je drevo brez korena, ki ima toliko listov, kot je objektov v $\mathcal{S}$, torej n, vsak od njih pa je označen z enim od objektov iz $\mathcal{S}$, pri čemer sta vsaka dva različna lista označena z različnima objektoma. Notranja vozlišča so prav tako označena in predstavljajo hipotetične prednike objektov iz množice $\mathcal{S}$ [3].

Lastnost c v takem drevesu T je *konveksna* (tudi: *kompatibilna*), če za vsak $a \in \mathcal{A}$, pri čemer je $\mathcal{A}$ množica vseh dovoljenih stanj, velja, da množica vozlišč v iz T, za katere je $c(s_v) = a$, tvori poddrevo drevesa T. Če je vsak $c \in \mathcal{C}$ v drevesu T konveksen, potem je I *popolna filogenija* in T njeno *popolno filogenetsko drevo*, kot je prikazano na sliki 5 desno. Velja tudi obratno; če za dano množico lastnosti $\mathcal{C}$ obstaja popolna filogenija, potem so lastnosti v $\mathcal{C}$ kompatibilne.

Problem popolne filogenije torej sprašuje, ali za dani množici objektov $\mathcal{S}$ in lastnosti $\mathcal{C}$ obstaja popolna filogenija, torej tako drevo T, v katerem so vsi $c \in \mathcal{C}$ kompatibilni [2].

Da so vsi objekti kompatibilni in je filogenija popolna, v njej ne sme priti do reverznih mutacij in vzporedne evolucije. Jasno je, da so take filogenije redkost, kadar gre za analizo biomolekularnih sekvenc (DNA in RNA), imajo pa svoj smisel pri raziskavah morfoloških značilnosti, kot je npr. prisotnost ali odsotnost hrbtenice [21].



Slika 5: Matrika s stanji lastnosti in primer popolne filogenije za to matriko

3.1 Popolna filogenija v splošnem

Problem popolne filogenije se v splošnem prevede na problem triangulacije obarvanih grafov, kot je v članku [6] dokazal Buneman.

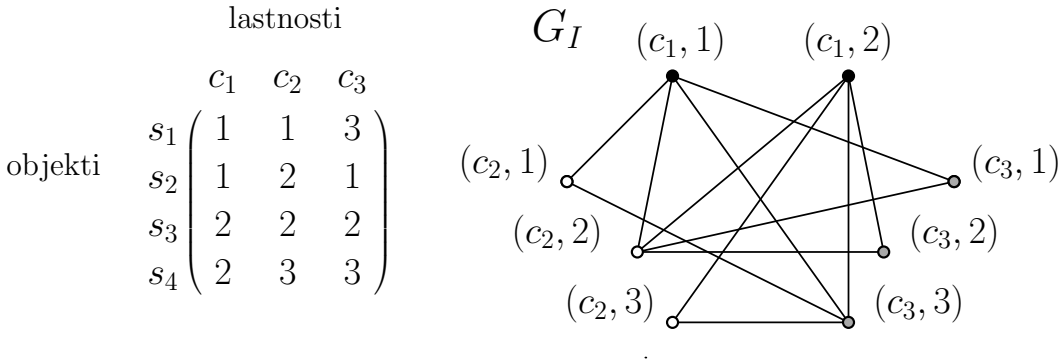
Definicija problema triangulacije obarvanih grafov je sledeča: zanima nas, ali lahko danemu grafu $G = (V, E)$, obarvanemu z barvanjem $c : V \to \{1, 2, \ldots, k\}$, dodamo povezave tako, da bo dobljeni graf tetiven, ne da bi med sabo povezali vozlišči enake barve. Če je to možno, rečemo, da smo graf G c-triangulirali, dobljenemu tetivnemu nadgrafu grafa G pa rečemo *c-triangulacija* [21].

Za obrazložitev zveze med problemoma popolne filogenije in triangulacije obarvanih grafov definirajmo najprej *particijski presečni graf* G_I na podatkih $I = (\mathcal{S}, \mathcal{C})$, pri čemer je $\mathcal{S}$ množica objektov in $\mathcal{C}$ množica njihovih lastnosti. Vozlišča v G_I so stanja lastnosti objektov iz $\mathcal{C}$ in sicer je (c, a) vozlišče, če je lastnost $c \in \mathcal{C}$ in je a stanje $c(s)$ lastnosti c za nek objekt $s \in \mathcal{S}$. Povezave v G_I so definirane sledeče: med (c_1, i) in (c_2, j), kjer je $c_1 \neq c_2$, obstaja povezava v grafu G_I če in samo če za nek s iz množice objektov $\mathcal{S}$ velja $c_1(s) = i$ in $c_2(s) = j$. V grafu G_I so torej povezana vozlišča, ki pripadajo istemu objektu, oziroma vozlišča, ki predstavljajo stanja posamezne lastnosti za določeni objekt s [2].

Graf G_I je pravilno obarvan s $|\mathcal{C}|$ barvami in vsebuje $|\mathcal{S}|$ klik. Z barvo i pobarvamo vozlišča, ki so v prvi koordinati označena s c_i, torej tista, ki predstavljajo stanja iste lastnosti. Jasno je, da me takimi vozlišči ne more biti povezave, ker bi to pomenilo, da ima neka lastnost določenega objekta dve različni stanji za isti objekt. Število klik je enako številu objektov, ker za vsak objekt povezave med vozlišči, ki predstavljajo stanje lastnosti za ta objekt, tvorijo kliko.

Primer obarvanega particijskega presečnega grafa je na sliki 6, število klik pa je lepše razvidno iz slike 7, ki predstavlja isti presečni graf kot slika 6.

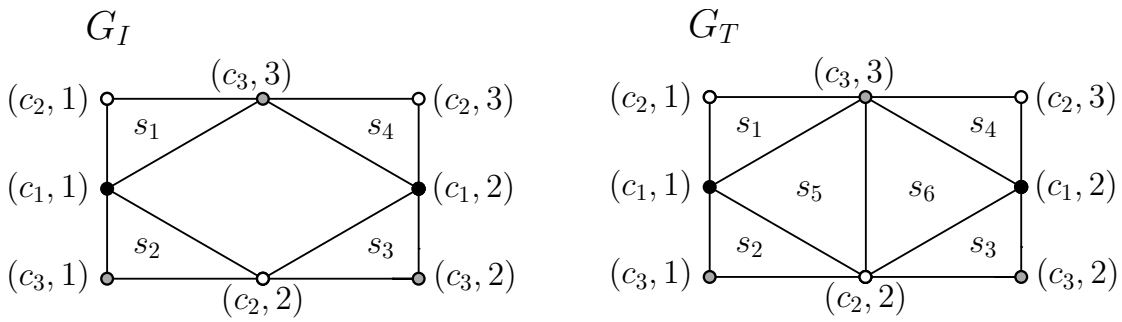
Lema 3.1. *Če za množico objektov $\mathcal{S}$ in množico njihovih lastnosti $\mathcal{C}$ obstaja popolna filogenija $I = (\mathcal{S}, \mathcal{C})$, potem je particijski presečni graf G_I moč c-triangulirati.*



Slika 6: Matrika s stanji lastnosti in njen particijski presečni graf

Particijski presečni graf, ki ga dobimo iz vhodnih podatkov $I = (\mathcal{S}, \mathcal{C})$, namreč ni nujno tetiven. Če graf G_I iz slike 6 narišemo malo drugače, dobimo graf G_I na sliki 7 levo, iz katerega so lepo razvidne klike, od katerih vsaka ustreza natanko enemu objektu iz $\mathcal{S}$. Graf G_I očitno ni tetiven, lahko pa ga c-trianguliramo, kot je definirano v problemu triangulacije obarvanih grafov. Z novo povezavo med vozliščema $(c_2, 2)$ in $(c_3, 3)$ dobimo tetivni nadgraf G_T, prikazan na sliki 7 desno.

Triangulacija, dobljena z dodajanjem povezave med vozliščema $(c_1, 1)$ in $(c_1, 2)$, ni c-triangulacija, saj definicija problema triangulacije obarvanih grafov ne dovoljuje dodajanja povezav med vozlišči iste barve, taka povezava bi namreč pomenila, da ima neki objekt lastnost c_1 v dveh različnih stanjih, kar je nemogoče. V našem primeru je torej graf G_T edina možna c-triangulacija grafa G_I, v splošnem pa jih lahko obstaja več.



Slika 7: c-triangulacija grafa G_I

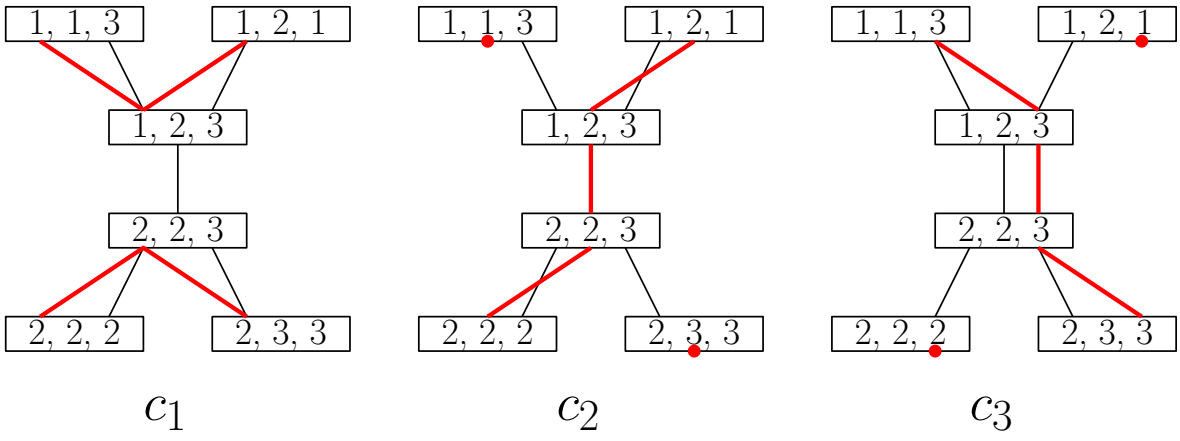
Lema 3.1 velja tudi v obratni smeri, in sicer kot posledica izrekov Bunemana in Gavrila (izrek 2.1 in izrek 2.4), ki skupaj tetivne grafe karakterizirata kot presečne grafe družine poddreves nekega drevesa T.

Posledica 3.2. *Če obstaja c-triangulacija grafa G_I, ki je particijski presečni graf za $I = (\mathcal{S}, \mathcal{C})$, pri čemer je $\mathcal{S}$ množica objektov in $\mathcal{C}$ množica njihovih lastnosti, potem za I obstaja popolna filogenija.*

Z uporabo izreka Bunemana in Gavrila (izrek 2.4) lahko iz grafa G_T na sliki 7 po postopku, ki je podrobno opisan v primeru 2.5 zgradimo drevo, prikazano na sliki 5 desno. Vozlišča drevesa namesto s kliko, kot smo to naredili v primeru 2.5, označimo z vozlišči posamezne klike.

Drevo ima 4 liste, ki ustrezajo objektom vhodne matrike s stanji lastnosti, notranji vozlišči $(1, 2, 3)$, $(2, 2, 3)$ pa predstavljata objekta, ki smo ju dobili s c-triangulacijo particijskega presečnega grafa G_I in ustrezata po c-triangulaciji nastalim klikam s_5 na vozliščih $(c_1, 1)$, $(c_2, 2)$, $(c_3, 3)$ ter s_6 na vozliščih $(c_1, 2)$, $(c_2, 2)$, $(c_3, 3)$ v grafu G_T na sliki 7.

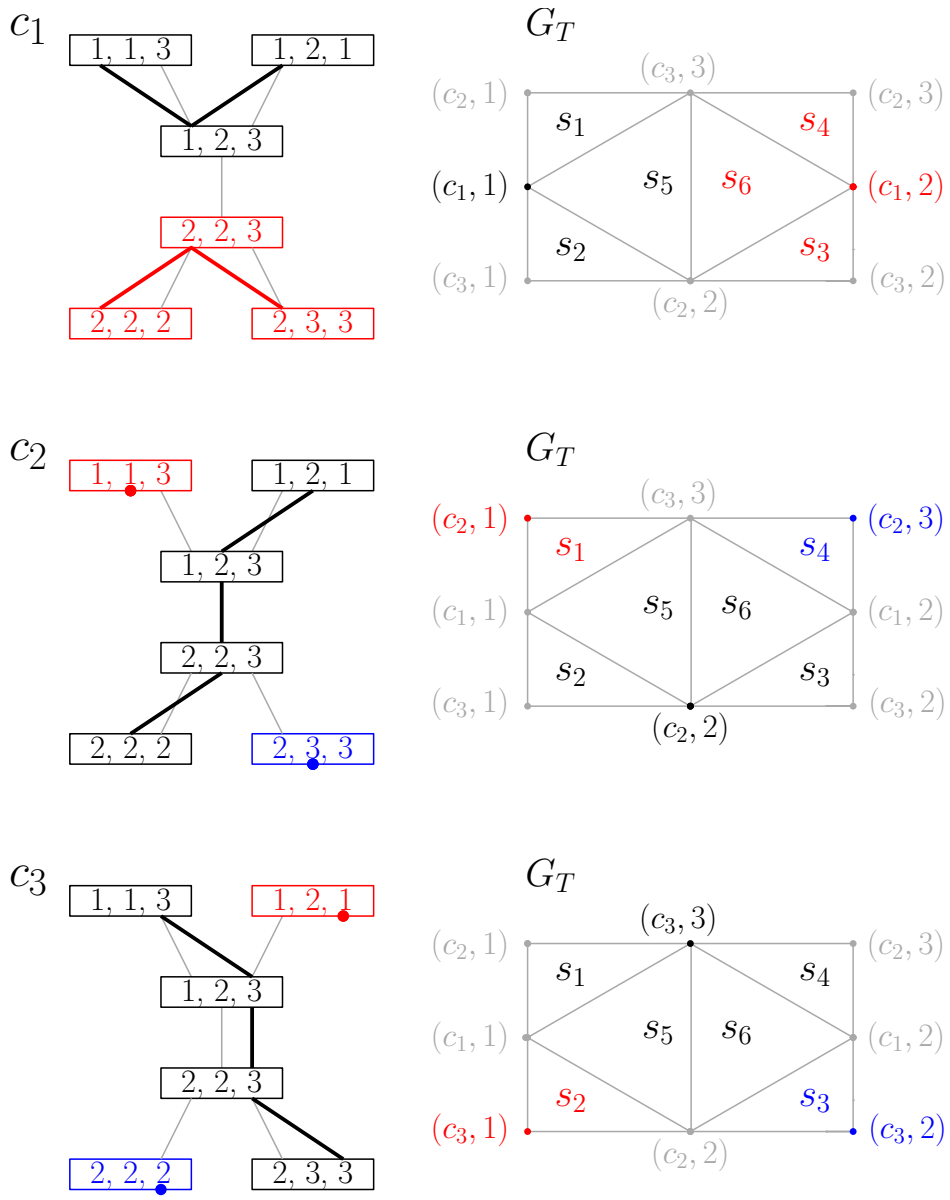
Preverimo še, ali so vse lastnosti c v drevesu T konveksne, torej da množica vozlišč v iz T, za vsako dovoljeno stanje lastnosti, tvori poddrevo drevesa T. Poddrevesa so označena na sliki 8 in sicer so zaradi večje preglednosti prikazana za vsako lastnost posebej. Razvidno je, da so vse lastnosti v drevesu res konveksne, iz česar sledi, da drevo na sliki 5, ki smo ga zgradili iz c-trianguliranega grafa G_T, ustreza popolni filogeniji.



Slika 8: Poddrevesa filogenetskega drevesa glede na stanja lastnosti

Spomnimo se še izreka Bunemana in Gavrila 2.1, ki pravi, da je presečni graf družine poddreves nekega drevesa T tetivni. Vidimo, da presečni graf poddreves iz slike 8 natanko ustreza tetivnemu grafu G_T. Vseh poddreves je osem, dve za lastnost c_1 ter po tri za lastnosti c_2 in c_3. Vsako od poddreves se ujema z eno točko grafa G_T, kot je razvidno iz slike 9.

Zgolj zaradi večje preglednosti so poddrevesa ponovno predstavljena za vsako lastnost posebej. Če pogledamo poddrevesi za lastnost c_1, je črno poddrevo na vozliščih,



Slika 9: Presečni graf poddreves ustreza tetivnemu grafu G_T

kjer ima lastnost c_1 stanje 1, to so $(1, 1, 3)$, $(1, 2, 3)$ in $(1, 2, 1)$, rdeče pa na vozliščih, kjer ima lastnost c_1 stanje 2, to so $(2, 2, 2)$, $(2, 2, 3)$ in $(2, 3, 3)$. Črno poddrevo ustreza točki $(c_1, 1)$ v grafu G_T, ki je vsebovana v klikah s_1, s_5 in s_2, te pa predstavljajo ravno vozlišča, ki jih črno poddrevo povezuje: klika s_1 vsebuje točke $(c_1, 1)$, $(c_2, 1)$ in $(c_3, 3)$ ter torej predstavlja vozlišče $(1, 1, 3)$, s_5 na točkah $(c_1, 1)$, $(c_2, 2)$ in $(c_3, 3)$ ustreza $(1, 2, 3)$, s_2 pa $(1, 2, 1)$. Podobno velja za točko $(c_1, 2)$, ki jo predstavlja rdeče poddrevo ter za vsa ostala poddrevesa lastnosti c_2 in c_3.

Tetivni graf G_T, ki je definiran kot presečni graf družine poddreves popolnega filogenetskega drevesa T je torej isti graf, kot ga dobimo po c-triangulaciji grafa G_I na podatkih $I = (\mathcal{S}, \mathcal{C})$, kjer $\mathcal{S}$ označuje množico objektov, $\mathcal{C}$ pa množico njihovih

lastnosti.

Izrek 3.3. *Za množico objektov $\mathcal{S}$ in množico njihovih lastnosti $\mathcal{C}$ obstaja popolna filogenija T natanko tedaj, ko je moč particijski presečni graf G_I, pri čemer je $I = (\mathcal{S}, \mathcal{C})$, c-triangulirati.*

Skica dokaza. ($\Rightarrow$) Iz popolne filogenije T definiramo G_T kot presečni graf družine poddreves, ki obstajajo po definiciji popolne filogenije kot poddrevesa $T_{v,a}$ popolnega filogenetskega drevesa T na množici vozlišč drevesa T, za katera velja $c(s_v) = a$, pri čemer c označuje lastnost, s objekt in a stanje lastnosti za objekt. Uporabimo izrek Bunemana in Gavrila (izrek 2.1), iz katerega sledi, da je G_T tetiven graf. Zagotovo velja, da $G_I \subseteq G_T$, saj je vsaka povezava v G_I tudi v G_T. Sledi torej, da je G_T c-triangulacija grafa G_I.

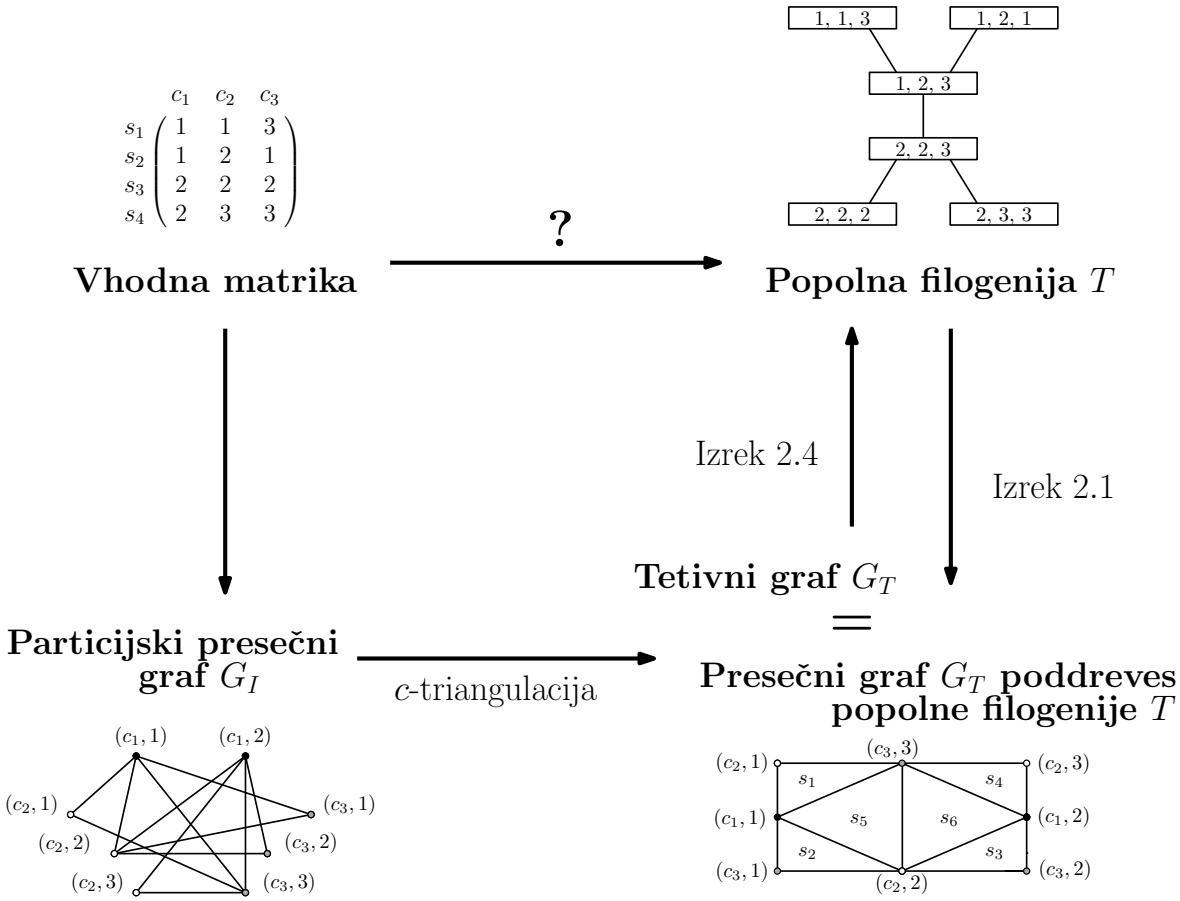
($\Leftarrow$) Če je graf G_I moč c-triangulirati, potem obstaja tak tetiven graf G_T, da velja $G_I \subseteq G_T$. Graf G_T je ravno c-triangulacija grafa G_I, iz česar z uporabo izreka Bunemana in Gavrila (izrek 2.4) sledi, da obstaja tako drevo T in družina njegovih poddreves, da je G_T ravno presečni graf te družine poddreves. S pomočjo te družine lahko definiramo popolno filogenijo. □

Kot smo omenili na začetku poglavja, se problem popolne filogenije reducira na problem triangulacije obarvanih grafov. Iz tega sledi zgornji izrek 3.3, ki pravi, da je vprašanje, ali za dane vhodne podatke obstaja popolna filogenija, ekvivalentno vprašanju, ali za particijski presečni graf na teh vhodnih podatkih obstaja c-triangulacija.

Na sliki 10 je z diagramom prikazana zgoraj opisana ekvivalenca med problemoma popolne filogenije in triangulacije obarvanih grafov.

Če na kratko povzamemo: zanima nas, ali za vhodno matriko, ki predstavlja stanja lastnosti določenih objektov, obstaja popolna filogenija. Iz matrike dobimo particijski presečni graf G_I, za katerega iz posledice 3.2 leme 3.1 velja, da popolna filogenija obstaja natanko tedaj, ko je moč ta particijski presečni graf G_I c-triangulirati. Po c-triangulaciji grafa G_I dobimo tetivni graf G_T, zanj pa po izreku 2.4 obstaja neko drevo T, hkrati pa z izrekom 2.1 iz popolne filogenije T dobimo tetivni graf G_T kot presečni graf družine poddreves tega drevesa T. Tetivni graf G_T, ki ga dobimo s c-triangulacijo particijskega presečnega grafa G_I, je torej hkrati tudi presečni graf družine poddreves popolne filogenije T, iz česar sledi, da je vprašanje, ali za vhodno matriko obstaja popolna filogenija, ekvivalentno vprašanju, ali je moč dani obarvani particijski presečni graf G_I triangulirati.

Problem triangulacije obarvanih grafov je NP-poln [3, 19]. Warnow je v doktorski disertaciji pokazala še, da sta problema popolne filogenije in triangulacije obarvanih



Slika 10: Diagram prevedbe problema popolne filogenije na problem triangulacije obarvanih grafov

grafov polinomsko ekvivalentna [22]. Iz tega sledi, da je tudi problem popolne filogenije v splošnem NP-poln.

Obstajajo pa posebni primeri vhodnih parametrov, za katere je problem triangulacije obarvanih grafov in s tem popolne filogenije rešljiv v polinomskem ali celo linearnem času. Popolna filogenija ima tri različne parametre: množico objektov $\mathcal{S}$, množico njihovih lastnosti $\mathcal{C}$ in množico stanj $\mathcal{A}$, ki jih te lastnosti lahko zavzamejo. Omenili bomo nekaj rešitev popolne filogenije za konstantno število lastnosti in konstantno število stanj.

3.1.1 Konstantno število lastnosti

Za konstantno število lastnosti $\mathcal{C}$ lahko vhodnim podatkom I priredimo particijski presečni graf G_I ter nato z uporabo izreka Bunemana in Gavrila (izrek 2.4) določimo, ali za I obstaja popolna filogenija.

Izrek 3.4 (Warnow [21]). *Za vhodne podatke $I = (\mathcal{S}, \mathcal{C})$, pri čemer je $|\mathcal{C}| = 2$, lahko*

obstoj popolne filogenije za I preverimo v času $O(|\mathcal{S}|)$.

Dokaz. Naj bo G' c-triangulacija grafa G_I, kjer je G_I particijski presečni graf. Graf G_I je obarvan z dvema barvama, ki ustrezata dvema lastnostma, torej je tudi G' obarvan z dvema barvama. Graf, obarvan z dvema barvama, je tetiven samo, če je acikličen. Torej mora biti G' acikličen, iz česar sledi, da je acikličen tudi G_I. Predpostavimo lahko tudi, da je $|V(G_I)| > |E(G_I)|$, saj bi bil v nasprotnem primeru G_I cikličen. Ker je G_I particijski presečni graf za dve lastnosti, je $|E(G_I)| = |\mathcal{S}|$, saj vsaka povezava v G_I ustreza objektu iz $\mathcal{S}$. Določanje grafa G_I porabi $O(|\mathcal{S}|)$ časa, določanje acikličnosti prav tako, torej algoritem porabi $O(|\mathcal{S}|)$ časa. □

Algoritmi za popolno filogenijo s tremi lastnostmi, ki prav temeljijo na tetivnih grafih in tečejo v linearnem času, so predstavljeni v člankih Kannana in Warnow [14] ter Bodlaenderja in Kloksa [4]. Obstajajo tudi polinomski algoritmi za omejeno število lastnosti. Eden takih algoritmov in njegova implementacija sta opisana v članku [18], kjer je pokazana zveza med k-obarljivimi grafi, za katere obstaja c-triangulacija in grafi, katerih drevesna širina je omejena s $k - 1$.

3.1.2 Konstantno število stanj

Pri bioloških podatkih imamo tipično opravka z manjšim ali vsaj omejenim številom možnih stanj, zato je popolna filogenija konstantnega števila stanj zelo uporabna tudi v praksi. Pri sekvenciranju DNA in RNA so za eno mesto na primer možna štiri različna stanja, ki ustrezajo nukleotidom [21].

Za binarno število stanj obstajajo polinomski in celo linearni algoritmi [10]. Popolna filogenija za dve različni stanji je podrobneje obravnavana v naslednjem razdelku.

Polinomski algoritem v času $O(n^2k)$, kjer je n število objektov in k število lastnosti za tri in štiri stanja, sta razvila Kannan in Warnow [15]. Da je problem popolne filogenije polinomsko rešljiv za poljubno omejeno število stanj, sta leta 1994 pokazala Agarwala in Fernández-Baca [1], hitrejši algoritem za ta problem pa sta kmalu zatem objavila Kannan in Warnow [16].

3.2 Popolna filogenija za binarne matrike

Binarna matrika je matrika, katere vrednosti so lahko samo 0 ali 1. Spet imamo množico $\mathcal{S}$ z n objekti in množico $\mathcal{C}$ z m lastnostmi, le da ima zdaj vsaka posamezna lastnost lahko le dve stanji – 0 ali 1. Zanima nas, ali je za dane objekte in njihove lastnosti, ki so predstavljeni v binarni matriki, mogoče zgraditi pripadajoče popolno

filogenetsko drevo. Pri tem je pred samo izgradnjo potrebno najprej preveriti, ali tako drevo obstaja.

Naj bo M binarna matrika velikosti $n \times m$, kjer so n objekti in m njihove pripadajoče lastnosti. Polje (i, j) v matriki M ima vrednost 1, če in samo če ima objekt i lastnost j.

Popolno filogenetsko drevo take matrike M je drevo s korenom T, za katerega predpostavimo, da koren drevesa nima nobene od m lastnosti, torej imajo vse njegove lastnosti vrednost 0, ter da se vsaka lastnost iz stanja 0 v stanje 1 v celotnem drevesu spremeni natanko enkrat in da se iz stanja 1 ne more povrniti v stanje 0. Velja, da vsakemu izmed n danih objektov pripada natanko en list v T in obratno, vsaki lastnosti m pripada natanko ena povezava v T, za katerikoli list w v drevesu T pa lastnosti po povezavah na poti od korena proti listu natančno opišejo vektor lastnosti, ki veljajo za objekt na listu w. Po zgornjih predpostavkah namreč v korenu dobimo vektor samih ničel, saj koren nima nobene od m lastnosti, na poti od korena do lista w pa lastnosti, ki so na povezavah te poti označene, dobijo vrednost 1.

Definicija 3.5. Za dano binarno matriko M stolpec i *vsebuje* stolpec j, če in samo če za vsako vrstico r velja, da je $M(r, i) \geq M(r, j)$.

Definicija 3.6. Za dano binarno matriko M je presek stolpcev i in j *prazen*, če za vsako vrstico r velja, da če je $M(r, i) = 1$, potem je $M(r, j) = 0$, in obratno, če je $M(r, j) = 1$, potem je $M(r, i) = 0$.

Naslednji izrek karakterizira, kdaj za dano matriko popolno filogenetsko drevo obstaja.

Izrek 3.7 (Gusfield [10])**.** *Popolno filogenetsko drevo $n \times m$ binarne matrike M obstaja, če in samo če za vsak par stolpcev C_i in C_j velja, da eden vsebuje drugega ali pa je njun presek prazen.*

Algoritem, narejen direktno po zgornjem izreku, potrebuje $O(mn^2)$ časa, da ugotovi, ali za dano matriko M obstaja popolno filogenetsko drevo. V splošnem pa vsak algoritem za ta problem potrebuje $\Omega(mn)$ časa, saj mora v najslabšem možnem primeru pregledati vse elemente matrike M. Dokaz tega dejstva je izpeljan s t.i. *metodo nasprotnika*, ki s sprotnim dodeljevanjem vrednosti še nepregledanim elementom matrike algoritem v izvajanju prisili, da pregleda vsa polja matrike M, preden se lahko odloči, ali popolno filogenetsko drevo obstaja ali ne. Metoda nasprotnika in njena implementacija sta podrobno opisani v članku [10].

Gusfield je z upoštevanjem izreka 3.7 prišel do algoritma 1, ki je opisan v nadaljevanju in obstoj popolnega filogenetskega drevesa preveri v času $O(nm)$ [10], obstaja pa še izboljšava na čas $O(k)$, pri čemer je k število enic v matriki M [2].

Algoritem 1: Preverjanje obstoja filogenetskega drevesa

Vhod: Binarna matrika M', ki jo dobimo po odstranitvi podvojenih stolpcev iz M

Izhod: Popolno filogenetsko drevo obstaja/Popolno filogenetsko drevo ne obstaja

```
za vsako polje M'(i,j)
    če M'(i,j) = 0 potem
        L(i,j) = " − " (prazno polje)
    če M'(i,j) = 1 potem
        če obstaja tak k < j, da velja M'(i,k) = 1 potem
            L(i,j) = k, za največji tak k
        sicer
            L(i,j) = 0
za vsak stolpec j
    L(j) = max_i L(i,j)
če za vse (i,j), za katere je M(i,j) = 1, velja L(i,j) = L(j) potem
    vrni Popolno filogenetsko drevo obstaja
sicer
    vrni Popolno filogenetsko drevo ne obstaja
```

Primer 3.8. Na stolpce v matriki M lahko gledamo kot na binarna števila z najpomembnejšim bitom v prvi vrstici.

$$\begin{array}{c} M \\ A \\ B \\ C \\ D \\ E \end{array} \begin{array}{c} \begin{array}{ccccc} m_1 & m_2 & m_3 & m_4 & m_5 \end{array} \\ \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

Uredimo jih od največjega do najmanjšega tako, da je največje število v prvem stolpcu. Izbrišemo morebitne podvojene stolpce in rezultat shranimo v novo matriko M'. Ustvarimo še prazno matriko L, ki je istih dimenzij kot M'.

$$\begin{array}{c} M' \\ A \\ B \\ C \\ D \\ E \end{array} \begin{array}{c} \begin{array}{ccccc} m_2 & m_1 & m_3 & m_5 & m_4 \end{array} \\ \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

Polja (i, j), na katerih ima M' vrednost 0, so v L prazna. Za vsako polje (i, j) z vrednostjo 1 v M', ima L tem polju vrednost, enako največjemu indeksu $k < j$, za katerega ima polje $M'(i, k)$ vrednost 1. Če tak indeks k ne obstaja, je $L(i, j)$ enak 0. V L torej zapišemo indeks, na katerem je v trenutni vrstici zadnja vrednost 1 v M' izključno s trenutno obravnavano.

$$\begin{array}{c} L \\ A \\ B \\ C \\ D \\ E \end{array} \begin{array}{c} \begin{array}{ccccc} l_1 & l_2 & l_3 & l_4 & l_5 \end{array} \\ \begin{pmatrix} 0 & 1 & - & - & - \\ - & - & 0 & - & - \\ 0 & 1 & - & 2 & - \\ - & - & 0 & - & 3 \\ 0 & - & - & - & - \end{pmatrix} \end{array}$$

Vsakemu stolpcu j nato priredimo največjo vrednost $L(i, j)$, ki jo označimo z $L(j)$. V našem primeru velja, da je $L(l_1) = 0$, $L(l_2) = 1$, $L(l_3) = 0$, $L(l_4) = 2$ in $L(l_5) = 3$. Preverimo, ali so vsi $L(i, j) = L(j)$, torej ali so vse številke v stolpcu enake največji vrednosti stolpca j. Ker to drži, potem popolno filogenetsko drevo za matriko M obstaja. ▲

Sledi še primer matrike, za katero popolno filogenetsko drevo ne obstaja.

Primer 3.9. Dana je matrika N.

$$\begin{array}{c} N \\ A \\ B \\ C \\ D \\ E \end{array} \begin{array}{c} \begin{array}{ccccc} n_1 & n_2 & n_3 & n_4 & n_5 \end{array} \\ \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \end{pmatrix} \end{array}$$

Po preureditvi stolpcev dobimo matriko M', ki ji, tako kot v prejšnjem primeru, priredimo matriko L.

$$
\begin{array}{c} N' \\ A \\ B \\ C \\ D \\ E \end{array}
\begin{array}{c}
\begin{array}{ccccc} n_1 & n_2 & n_3 & n_4 & n_5 \end{array} \\
\begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \end{pmatrix}
\end{array}
$$

$$
\begin{array}{c} L \\ A \\ B \\ C \\ D \\ E \end{array}
\begin{array}{c}
\begin{array}{ccccc} l_1 & l_2 & l_3 & l_4 & l_5 \end{array} \\
\begin{pmatrix} 0 & 1 & - & - & - \\ - & - & 0 & 3 & - \\ 0 & 1 & 2 & - & - \\ - & - & - & 0 & 4 \\ 0 & - & 1 & - & - \end{pmatrix}
\end{array}
$$

Vrednosti $L(j)$ so $L(l_1) = 0$, $L(l_2) = 1$, $L(l_3) = 2$, $L(l_4) = 3$, $L(l_5) = 4$, vendar ni res, da je $L(i, j) = L(j)$ za vsako neprazno polje $L(i, j)$. Za $L(l_3) = 2$ vidimo, da je $L(B, l_3) = 0$ in $L(E, l_3) = 1$, za $L(l_4) = 3$ pa je $L(D, l_4) = 0$. Popolno filogenetsko drevo matrike N ne obstaja. ▲

Ko smo se prepričali, da popolno filogenetsko drevo matrike M iz primera 3.8 obstaja, ga lahko zgradimo po spodnjem algoritmu 2. Sam postopek izgradnje drevesa je opisan v naslednjem primeru.

Primer 3.10. Za vsak stolpec j matrike M' ustvarimo vozlišče n_j, kot je prikazano na primeru na sliki 11 zgoraj levo. V naslednjem koraku dodamo še korensko vozlišče r. Na poljih, kjer je v matriki $L(j) = 0$, povežemo vozlišče n_j s korenom r. Na poljih, kjer je $L(j) > 0$, povežemo n_j in $n_{L(j)}$. Povezave označimo z indeksom j ter indeksi stolpcev, ki so enaki j-temu stolpcu in so bili v prvem koraku izbrisani. Dobimo T_1.

V drugem delu algoritma gledamo vrstice matrike M'. Za vsako vrstico i je c_i največji indeks, za katerega velja, da ima polje (i, c_i) v matriki M' vrednost 1, torej je c_i indeks zadnje vrednosti 1 posamezne vrstice. V drevesu najdemo povezavo, označeno s c_i. Če povezava vodi do lista, na list pripnemo i, sicer pa iz vozlišča ustvarimo novo povezavo in i pripnemo na na novo ustvarjeni list.

V prvi iteraciji se osredotočimo na vrstico A v matriki M'. Indeks c_i je v tem primeru 2, saj je zadnja enica v vrstici A na drugem mestu. V drevesu T_1 najdemo povezavo, označeno z 2, in iz njenega konca ustvarimo novo povezavo, na katero pripnemo A, ter s tem dobimo drevo T_2. V naslednji vrstici B je zadnja enica na tretjem mestu, v drevesu najdemo ustrezno povezavo, ustvarimo novo in na list pripnemo B, kot je prikazano v T_3. V naslednji iteraciji vstavimo C, saj je v vrstici C zadnji stolpec z vrednostjo 1 na indeksu 4, povezava 4 v drevesu pa vodi do lista. Rezultat je drevo

T_4. Na isti način vstavljamo, dokler ne dobimo popolnega filogenetskega drevesa, kot je prikazano na T_5 na sliki 11. ▲

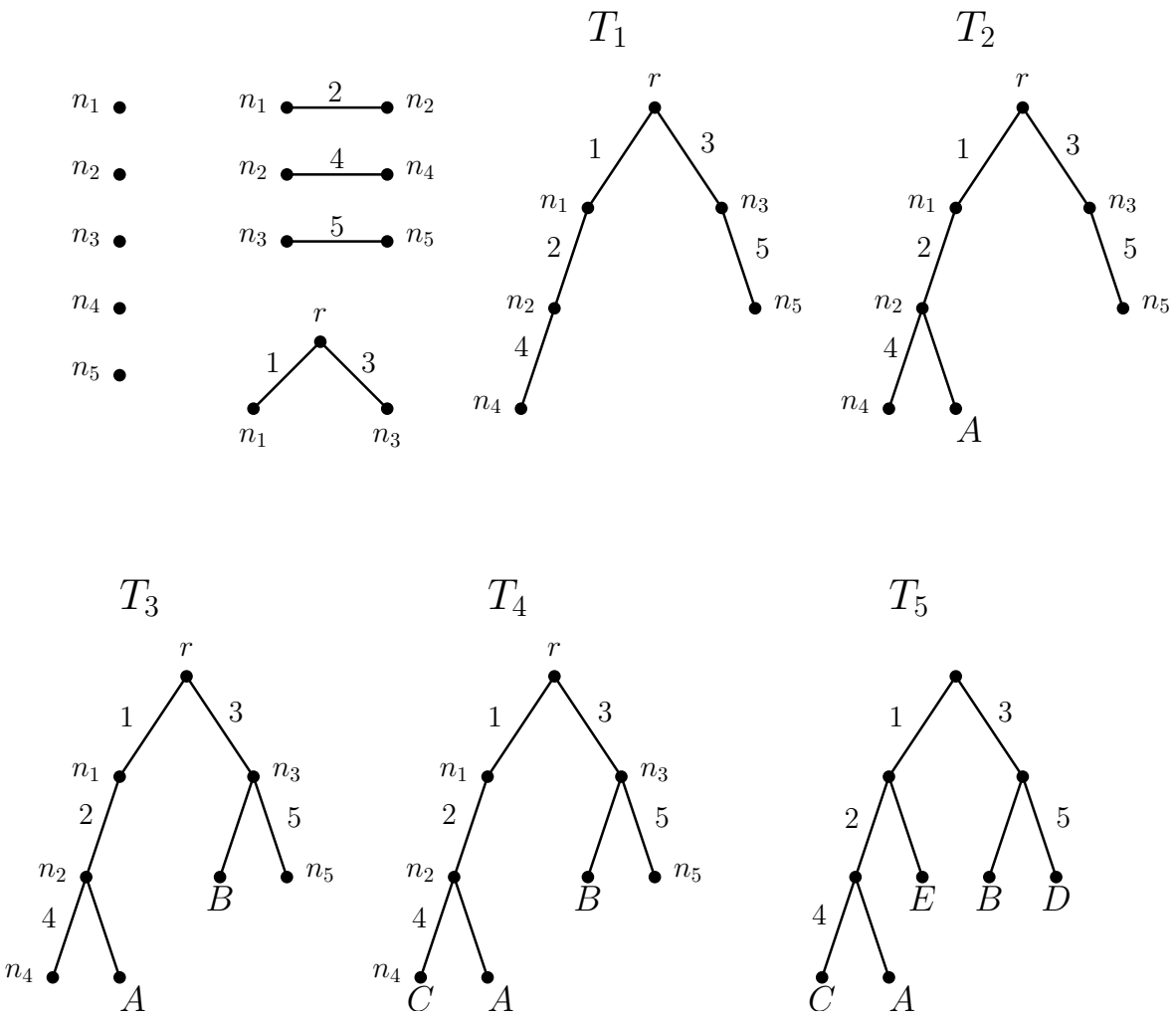
Algoritem 2: Izgradnja popolnega filogenetskega drevesa

Vhod: Binarna matrika M'

Izhod: Popolno filogenetsko drevo T

```
za vsak stolpec j iz M'
    ustvari vozlišče n_j
ustvari korensko vozlišče r
za vsako vozlišče n_j
    če L(j) > 0 potem
        poveži n_{L(j)} z n_j in označi povezavo z j in z indeksi vseh stolpcev matrike
        M, ki so enaki j-temu stolpcu
    če L(j) = 0 potem
        poveži r z n_j in označi povezavo z j in z indeksi vseh stolpcev matrike
        M, ki so enaki j-temu stolpcu
za vsako vrstico i
    c_i je največji indeks, tako da velja, da M'(i, c_i) = 1
za vsako vrstico i
    najdi povezavo, označeno s c_i
    če je konec iskane povezave list potem
        prilepi i na list
    sicer
        ustvari novo povezavo iz zadnjega vozlišča in prilepi i na nov list
vrni T
```

Oba zgoraj opisana algoritma, algoritem 1 za preveritev obstoja popolnega filogenetskega drevesa in algoritem 2 za izgradnjo popolnega filogenetskega drevesa, je možno implementirati v času $O(nm)$ [10]. V istem članku najdemo tudi dokaz pravilnosti obeh algoritmov.



Slika 11: Gradnja popolnega filogenetskega drevesa

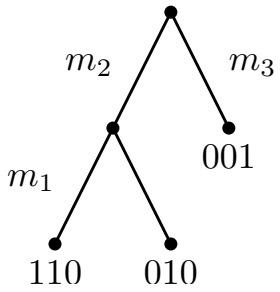
4 Problem mešane popolne filogenije

V tem poglavju so povzete ugotovitve iz članka Hajirasoulihe in Raphaela [11], v katerem je formuliran problem *mešane popolne filogenije*, optimizacijska različica problema in prevedba problema na problem barvanja grafov. Opisan je tudi njun algoritem, ki naj bi s pomočjo barvanja grafov rešil optimizacijsko verzijo problema mešane popolne filogenije, a se je žal izkazal za napačnega. Tako pravilno kot nepravilno delovanje algoritma je predstavljeno in podrobno opisano na primerih.

Opravka imamo z modeliranjem mutacijskega procesa znotraj tumorja in rekonstrukcijo zgodovine somatskih mutacij. Na problem rekonstrukcije somatskih mutacij lahko gledamo kot na problem filogenije, v kateri so objekti posamezne celice tumorja, lastnosti pa označujejo prisotnost oziroma odsotnost določenih mutacij. Predpostavimo lahko, da lastnosti zavzamejo samo dve različni stanji, in sicer 0 za normalno celico in 1 za mutirano celico, in da na vsaki poziciji vzdolž genoma do mutacije pride samo enkrat. Ob teh predpostavkah mutacijski proces tumorja ustreza popolni filogeniji, konstrukcija drevesa za popolne filogenije pa je možna, kot smo to videli v poglavju 3.

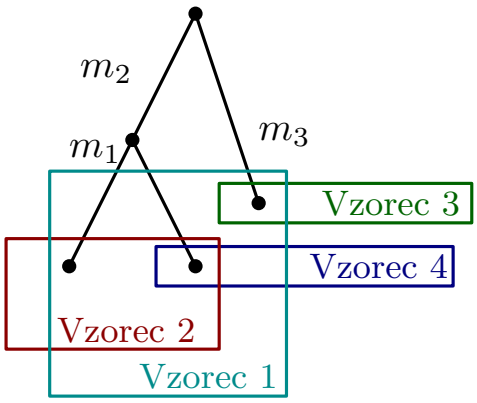
Rezultat sekvenciranja m pozicij genoma za n vzorcev tumorja, pri čemer ima lahko vsaka pozicija dve različni stanji, lahko predstavimo z $n \times m$ binarno matriko M, v kateri je vsak vzorec predstavljen z eno vrstico, vsaka lastnost pa z enim stolpcem. Polje (i, j) v matriki M ima vrednost 1, če je v lastnosti j vzorca i prisotna mutacija. Če vsak vzorec vsebuje samo eno celico tumorja in če predpostavimo, da se mutacija na določeni poziciji zgodi največ enkrat ter je predhodno stanje predstavljeno z genomom brez mutacij, ki ima v matriki vse vrednosti enake 0, potem so pogoji popolne filogenije izpolnjeni. V pripadajočem popolnem filogenetskem drevesu vsak list predstavlja celico, na povezavah pa so označene lastnosti, oziroma mutacije, ki se na tistem mestu spremenijo iz stanja 0 v stanje 1. Primer takega drevesa popolne filogenije je prikazan na sliki 12. Na povezavah so označene mutacije m_1, m_2, m_3, na listih pa zaporedja mutiranih celic.

Študije rakavega tkiva se v resnici ne osredotočajo na mutacije v posameznih celicah, ampak sekvencirajo DNA iz vzorca tumorja, ki vsebuje na tisoče ali celo milijone celic. Rekonstrukcija zgodovine takega vzorca se razlikuje od splošnega filogenetskega pro-



Slika 12: Popolna filogenija treh mutacij

blema, saj na voljo nimamo podatkov o posameznih objektih, ampak le o skupkih več objektov. V pripadajočem popolnem filogenetskem drevesu se ne smemo osredotočati na posamezne liste, ki predstavljajo celice, ampak na množice listov, ki ustrezajo posameznemu vzorcu. Na drevesu na sliki 13 so tako predstavljeni štirje vzorci, od katerih je vsak sestavljen iz enega ali več listov, torej iz ene ali več celic. Vzorec 1 ustreza zaporedju 111, saj so v njem prisotne vse tri mutacije m_1, m_2 in m_3, vzorec 2 zaporedju 110, saj vsebuje lista do katerih vodijo povezave, označene z mutacijama m_1 in m_2, vzorec 3 zaporedju 001 in vzorec 4 zaporedju 010.



Slika 13: Filogenija štirih vzorcev

Problem mešane popolne filogenije je razcep teh skupkov na posamezne objekte ter rekonstrukcija mutacijske zgodovine z uporabo modela popolne filogenije.

4.1 Minimalen razcep vrstic

Pred formulacijo problema minimalnega razcepa vrstic je potrebna definicija konfliktnih stolpcev v matriki.

Definicija 4.1. Stolpca C_i in C_j v binarni matriki M sta *konfliktna*, če obstajajo take

vrstice r_1, r_2, r_3 v M, katerih polja (i, j) (tj., v stolpcih C_i in C_j) zavzamejo zaporedoma vrednosti $(1, 1)$, $(1, 0)$ in $(0, 1)$.

Definicija 4.2. Matrika je *brez konfliktov*, če ne vsebuje nobenega para konfliktnih stolpcev.

Na tem mestu je nujno omeniti zvezo zgornje definicije z izrekom 3.7 iz prejšnjega poglavja. Matrika je namreč brez konfliktov natanko tedaj, ko je pogoj iz prej omenjenega izreka izpolnjen, torej ko za vsak par stolpcev velja, da eden vsebuje drugega ali pa je njun presek prazen. Iz tega sledi, da je odsotnost konfliktov potreben in zadosten pogoj za obstoj popolnega filogenetskega drevesa.

Predpostavili smo, da celice tumorja ustrezajo popolni filogeniji. Če predpostavimo še, da do napak v podatkih ne prihaja, potem so morebitni konflikti, ki se pojavijo v matriki M, lahko zgolj posledica dejstva, da vsaka vrstica v matriki predstavlja vzorec, ki ne vsebuje ene same celice, ampak skupek celic. Problem *mešane popolne filogenije* je razcepiti take vrstice tako, da bo vsaka predstavljala samo eno celico in da posledično v matriki, ki jo po takem razcepu dobimo, ne bo nobenega konflikta.

Definicija 4.3. Za dano vrstico r v matriki mutacij M definiramo *razcep vrstice* r kot poljubno tako množico vrstic $\{r'_1, r'_2, \ldots, r'_k\}$ (ne nujno v matriki M), da imajo za vsak i, kjer je i-ta pozicija v r enaka 0, vse vrstice r'_j na tej poziciji vrednost 0, za vsak i, kjer je i-ta pozicija v r enaka 1, pa ima vsaj ena od vrstic r'_j, na tej poziciji vrednost 1.

Problem mešane popolne filogenije je sledeč: za dano binarno matriko M z n vrsticami in m stolpci poišči binarno matriko M' brez konfliktov z n' vrsticami in m stolpci tako, da je M' brez konfliktov in da za vsako vrstico r v M obstaja $k = k_r$ vrstic $r'_1, r'_2, \ldots, r'_k$ v M', ki tvorijo razcep vrstice r.

Z drugimi besedami, za dano matriko mutacij M je naš cilj najti razcep vrstic, ki M pretvori v matriko brez konfliktov M', ki ustreza predpostavkam popolne filogenije in za katero je mogoče zgraditi popolno filogenetsko drevo. Razcep vrstic loči skupke celic v vzorcu na posamezne celice, ki so v drevesu predstavljene z listi. Tak razcep vrstic je seveda vedno mogoč z uporabo identične matrike za matriko M'.

Za razcep vrstice r na k vrstic definiramo še cenovno funkcijo $\gamma(M, r) = k - 1$, ki predstavlja število dodanih vrstic v matriki M' glede na M.

Oglejmo si primer razcepa vrstic konfliktne matrike ter njene cenovne funkcije. Na tem mestu je potrebno omeniti še, da je pri dodajanju vrstic treba paziti, da ne ustvarjamo novih konfliktov. Kot bomo videli v nadaljevanju, je namreč v splošnem ustvarjanje novih konfliktov možno, več o tem pa je napisano v [11, Observation 2].

Primer 4.4. Dana je matrika M.

$$\begin{array}{c} M \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{ccc} m_1 & m_2 & m_3 \end{array} \\ \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \end{array}$$

Stolpca m_2 in m_3 sta konfliktna. Po razcepu vrstic iz prve vrstice ustvarimo dve novi vrstici. Rezultat je matrika M', ki je brez konfliktov in ustreza popolni filogeniji.

$$\begin{array}{c} M' \\ A_1 \\ A_2 \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{ccc} m_1 & m_2 & m_3 \end{array} \\ \begin{pmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \end{array}$$

Vrednost funkcije $\gamma(M, A)$ je 1, saj smo glede na matriko M po razcepu dobili eno dodatno vrstico. ▲

Eden izmed načinov razcepa vrstic je, da vsaki vrstici r matrike M priredimo graf $G_{M,r}$, v katerem vozlišča predstavljajo polja, na katerih ima r vrednost 1. Vozlišči v grafu $G_{M,r}$ sta povezani s povezavo, če sta njuna stolpca v M v konfliktu. Da dobimo matriko brez konfliktov, moramo narediti razcep vrstic za vsako vrstico r, za katero je graf $G_{M,r}$ neprazen, torej je v njem vsaj ena povezava. Vsaka vrstica r mora biti zamenjana z vsaj $\chi(G_{M,r})$ vrsticami, pri čemer je $\chi(G)$ kromatično število grafa G.

Lema 4.5. *Da mutacijsko matriko M z n vrsticami in m stolpci z razcepom vrstic prevedemo v matriko brez konfliktov M', mora biti vsaka vrstica r v M zamenjana z vsaj $\chi(G_{M,r})$ vrsticami, kjer je $G_{M,r}$ graf vrstice r, kot smo ga definirali zgoraj.*

Problem minimalnega razcepa vrstic je sledeč: Za dano matriko mutacij M naredi razcep vrstic na dani podmnožici vrstic $\mathcal{S}$ matrike M tako, da bo nova matrika brez konfliktov, vrednost vsote $\sum_{r \in \mathcal{S}} \gamma(M, r)$ pa bo najmanjša možna.

Če podmnožica vrstic $\mathcal{S}$ ni podana kot del vhodnih podatkov, jo lahko z uporabo leme 4.5 izračunamo iz matrike M, in sicer tako, da v $\mathcal{S}$ damo vsako vrstico r, za katero je $\chi(G_{M,r}) > 1$.

V članku Hajirasoulihe in Raphaela (Theorem 4) najdemo dokaz o NP-polnosti problema minimalnega razcepa vrstic, ki temelji na trditvi, da lahko vsakemu enostavnemu neusmerjenemu grafu G priredimo binarno matriko M, katere prva vrstica,

označimo jo z r, vsebuje same vrednosti 1 in katere graf $G_{M,r}$ natanko ustreza grafu G. Podan je način konstrukcije take matrike, s katero je nato z uporabo leme 4.5 prikazana NP-težkost problema iskanja minimalnega števila novih vrstic.

Izkazalo se je, da takega grafa G iz poljubne matrike ne moremo konstruirati na predstavljeni način in dokaz je bil pred kratkim ovržen v članku [13]. V njem je podan nov dokaz NP-težkosti problema in formulirana še odločitvena različica problema minimalnega razcepa vrstic z dokazom njene NP-polnosti.

4.2 Algoritem za problem mešane popolne filogenije

Predstavljen bo algoritem za problem minimalnega razcepa vrstic, ki doseže spodnjo mejo, podano v lemi 4.5, ki pravi, da moramo za prevedbo konfliktne matrike M v matriko brez konfliktov M' vsako vrstico r zamenjati z vsaj $\chi(G_{M,r})$ vrsticami.

Pred nadaljevanjem se spomnimo definicije 3.5, v kateri je definirano, da stolpec i binarne matrike M *vsebuje* stolpec j, če in samo če za vsako vrstico k velja, da je $M_{k,i} \geq M_{k,j}$.

Definicija 4.6. Za dano binarno matriko M je njen *graf vsebovanja* H_M usmerjeni graf, katerega vozlišča ustrezajo stolpcem v M in obstaja usmerjena povezava $i \to j$, če in samo če stolpec j vsebuje stolpec i.

Če predpostavimo, da matrika nima parov identičnih stolpcev, velja, da je graf H_M acikličen digraf. Dani algoritem za razreševanje konfliktov v matriki M razrešuje konflikte z uporabo problema barvanja grafov, pri čemer za iskanje in razreševanje novih konfliktov po razcepu vseh vrstic preveri graf H_M.

Stolpce matrike M permutiramo tako, da so vozlišča grafa H_M *topološko* urejena. Vozlišče i je pred vozliščem j, če iz j ne moremo nazaj v i (po usmerjeni poti); ta pogoj velja za vse pare vozlišč (i, j). Obratno ne velja nujno – iz vozlišča i lahko obstaja pot do vozlišča j, ali pa tudi ne.

Naj bo $k = \chi(G_{M,r})$, torej kromatično število grafa konfliktov vrstice r, ki smo ga definirali proti koncu prejšnjega podpoglavja. Vozliščem v $G_{M,r}$ dodelimo k barv tako, da sta vozlišči, katerih stolpca sta v konfliktu, različne barve. Nato zamenjamo vrstico r v matriki M z natanko k vrsticami na sledeči način: v i-ti vrstici med temi k vrsticami, pri čemer velja, da $1 \leq i \leq k$, določimo vrednost 1 na polja stolpcev, ki ustrezajo tistim vozliščem, ki so v grafu $G_{M,r}$ obarvani z i. Ostala polja imajo vrednost 0.

Vstavljanje vrednosti 0 na ostala polja lahko privede do novih konfliktov. Če namreč obstaja tak par stolpcev m_1 in m_2, da v digrafu H_M iz vozlišča m_1 vodi povezava do

vozlišča m_2 (stolpec m_2 torej vsebuje stolpec m_1), v novi vrstici, ki jo označimo z r'_1 pa ima stolpec m_1 vrednost 1, stolpec m_2 pa vrednost 0, lahko dobimo nov konflikt.

Nove konflikte odpravimo s preverjanjem digrafa H_M po razcepu vseh vrstic. Pri tem uporabimo algoritem *pregleda digrafa v globino*. Digraf H_M ni nujno povezan, zato je potrebno ponoviti pregled v globino iz vsake komponente pripadajočega neusmerjenega grafa. Vrstni red izhodiščnih vozlišč je enak vrstnemu redu topološko urejenih stolpcev v matriki. Začnemo torej v vozlišču, ki pripada levemu stolpcu. Označimo ga kot obiskanega in se pomaknemo do njegovega poljubnega soseda. Postopek ponavljamo od trenutnega vozlišča, dokler ne pridemo do vozlišča, ki je brez sosedov, oziroma so vsi njegovi sosedje že obiskani. Takrat se vrnemo po isti poti in obiščemo prvega naslednjega še neobiskanega soseda nekega prejšnjega vozlišča. Ko so vsa vozlišča označena kot obiskana, smo pregledali vsa vozlišča v digrafu H_m, do katerih obstaja pot iz izhodiščnega vozlišča. Algoritem ponovimo iz vozlišča, ki pripada naslednjemu stolpcu matrike. Nadaljujemo, dokler ne preverimo digrafa v globino iz vsakega vozlišča kot izhodiščnega. Če v kateremkoli koraku preverjanja na poti po grafu najdemo povezavo, ki v določeni vrstici povezuje vrednost v matriki 1 z 0, spremenimo vrednost ustreznega polja iz 0 v 1.

Opisani algoritem temelji na pravilnem barvanju vozlišč grafa in izkazalo se je, da je barvanje grafa $G_{M,r}$ polinomski problem [13]. Če pa je za vsako vrstico r že dano pravilno barvanje grafa $G_{M,r}$, je algoritem zelo hiter in potrebuje $O(n(n+m))$ časa, kjer je n število vrstic in m število stolpcev.

Oglejmo si zdaj primer delovanja algoritma na matriki, ki privede do pravilnega rezultata.

Primer 4.7. Dana je matrika M s štirimi konflikti: stolpec m_3 je v konfliktu s stolpci m_1, m_2 in m_4, četrti konflikt pa je med stolpcema m_1 in m_4.

$$\begin{array}{c} M \\ A \\ B \\ C \\ D \\ E \end{array} \begin{array}{c} \begin{array}{cccc} m_1 & m_2 & m_3 & m_4 \end{array} \\ \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \end{array}$$

Graf vsebovanja H_M za matriko M je prikazan na sliki 14. Vozlišča v grafu predstavljajo posamezne stolpce matrike m. Ker stolpec m_2 vsebuje stolpca m_1 in m_4, graf vsebuje usmerjeni povezavi iz m_1 in m_4 do m_2.

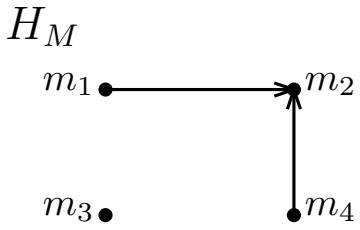
Če stolpce matrike M permutirano glede na topološko urejenost grafa H_M, dobimo sledečo matriko.

Algoritem 3: Algoritem za problem mešane popolne filogenije

Vhod: Binarna matrika M s permutiranimi stolpci, aciklični digraf vsebovanja H_M, grafi konfliktov $G_{M,r}$

Izhod: Binarna matrika M' brez konfliktov

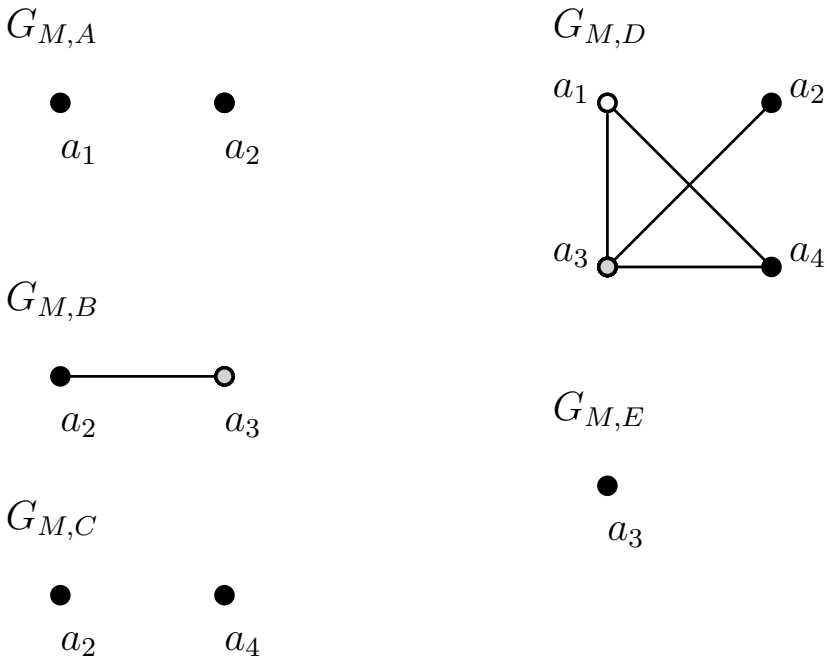
```
1  M = M'
2  za vsako vrstico r iz M
3      Naj bo c_r : V(G_{M,r}) → {1, ..., χ(G_{M,r})} k-barvanje grafa G_{M,r}, kjer je
       k = χ(G_{M,r})
4      v M' zamenjamo r s k vrsticami r_1, ..., r_k
5      za vsako novo vrstico r_i, kjer je i ∈ {1, ..., k}
6          za vsako vozlišče v grafa G_{M,r}
7              če c(v) = i potem
8                  M'(r_i, v) = 1
9              sicer
10                 M'(r_i, v) = 0
11 za vsak stolpec m iz M' od leve proti desni
12     preglej graf H_M v globino iz vozlišča m
13     če obstaja povezava v H_M iz m_i do m_j in velja, da v neki vrstici r:
       M'(r, m_i) = 1 in M'(r, m_j) = 0 potem
14         M'(r, m_j) = 1
15 vrni M'
```



Slika 14: Graf vsebovanja H_M

$$M \begin{matrix} \\ A \\ B \\ C \\ D \\ E \end{matrix} \begin{matrix} m_1 & m_4 & m_2 & m_3 \end{matrix} \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Pravilnih topoloških urejanj je v tem primeru več (vrstni red stolpcev bi bil lahko na primer tudi m_3, m_1, m_4, m_2), pomembno je le, da sta stolpca m_1 in m_4 v matriki pred stolpcem m_2.



Slika 15: Obarvani grafi G_M konfliktov posameznih vrstic

Graf konfliktov $G_{M,A}$, prikazan na sliki 15, ima dve vozlišči, a_1 in a_2, saj ima vrstica A vrednost 1 v stolpcih m_1 in m_2. Graf $G_{M,A}$ je brez povezav, saj med m_1 in m_2 ni konflikta. Obarvamo ga lahko z eno barvo in razcep vrstice A ni potreben.

Graf $G_{M,B}$ ima prav tako dve vozlišči, a_2 in a_3. Vozlišči a_2 in a_3 sta povezani, ker sta njuna pripadajoča stolpca m_2 in m_3 v konfliktu. Povezani vozlišči sta obarvani z različnima barvama, kromatično število k grafa $G_{M,B}$ je tako 2. Pobarvan graf $G_{M,B}$ je prikazan na sliki 15. Naj bo barva številka 1 črna, barva številka 2 pa siva.

Vrstico B v matriki M zamenjamo s k vrsticami, pri čemer je $k = 2$. V prvi novi vrstici B_1 pripišemo vrednost 1 na polja stolpcev, ki ustrezajo črnim vozliščem (rekli smo, da je barva številka 1 črna), torej v tretji stolpec m_2. V vrstici B_2 pripišemo vrednost 1 na polje stolpca, ki ustreza sivemu vozlišču, torej v četrti stolpec m_3. Preostali stolpci imajo v teh dveh vrsticah vrednost 0. Dobimo matriko M_1.

$$
\begin{array}{c} M_1 \\ A \\ B_1 \\ B_2 \\ C \\ D \\ E \end{array}
\begin{array}{c} \begin{array}{cccc} m_1 & m_4 & m_2 & m_3 \end{array} \\
\begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{array}
$$

Graf $G_{M,C}$ ima dve vozlišči, a_2 in a_4. Vrstici m_2 in m_4 v matriki nista v konfliktu, razcep vrstice ni potreben in nadaljujemo lahko z razcepom vrstice D.

$$
\begin{array}{c} M_2 \\ A \\ B_1 \\ B_2 \\ C \\ D \\ E \end{array}
\begin{array}{c} \begin{array}{cccc} m_1 & m_4 & m_2 & m_3 \end{array} \\
\begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{array}
$$

Graf $G_{M,D}$ ima štiri vozlišča, a_1, a_2, a_3 in a_4. Stolpec m_3 je v konfliktu s stolpci m_1, m_2 in m_4, zato je vozlišče a_3 povezano z vozlišči a_1, a_2 in a_4. Zaradi konflikta med m_1 in m_4 sta povezani še vozlišči a_1 in a_4. Za barvanje grafa $G_{M,D}$ potrebujemo tri barve, kot je prikazano na sliki 15. Naj bo barva številka 1 črna, barva številka 2 siva, barva številka 3 pa bela.

Vrstico D v matriki M_2 zamenjamo s k vrsticami, pri čemer je $k = 3$ kromatično število grafa $G_{M,D}$. V prvi novi vrstici D_1 pripišemo vrednost 1 na polja stolpcev, ki ustrezajo črnima vozliščema, torej v drugi (m_4) in tretji (m_2) stolpec. V drugi novi vrstici D_2 je vrednost 1 na polju stolpca, ki ustreza sivemu vozlišču, torej v četrtem stolpcu m_3. V tretji novi vrstici D_3 je vrednost 1 na polju stolpca, ki ustreza belemu vozlišču, torej v prvem stolpcu m_1. Preostali stolpci imajo v teh treh vrsticah vrednost 0. Dobimo matriko M_3.

$$
\begin{array}{c} M_3 \\ A \\ B_1 \\ B_2 \\ C \\ D_1 \\ D_2 \\ D_3 \\ E \end{array}
\begin{array}{c} \begin{array}{cccc} m_1 & m_4 & m_2 & m_3 \end{array} \\ \left(\begin{array}{cccc} 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{array} \right) \end{array}
$$

Graf $G_{M,E}$ na sliki 15 ima samo eno vozlišče, a_3, ki ustreza vrednosti 1 v stolpcu m_3. Za barvanje grafa potrebujemo eno samo barvo in razcep vrstice ni potreben.

Po končanem razcepu vrstic moramo poiskati in preveriti možne nove konflikte, ki so se med razcepom lahko pojavili v matriki. Pri preverjanju si pomagamo z grafom vsebovanja H_M in sicer začnemo iz vozlišča, ki predstavlja prvi stolpec matrike, v našem primeru je to m_1, iz katerega v grafu H_M vodi usmerjena povezava do m_2. To pomeni, da mora v vsaki vrstici, kjer ima m_1 vrednost 1, po definiciji vsebovanih stolpcev tudi m_2 imeti vrednost 1. Stolpec m_1 ima vrednosti 1 v vrsticah A in D_3, m_2 pa ima v vrstici A vrednost 1, v vrstici D_3 pa vrednost 0. Ustvarili smo torej nov konflikt med vrsticama m_1 in m_2, ki ga rešimo tako, da v vrstici D_3 stolpca m_2 spremenimo vrednost iz 0 v 1. Dobimo matriko M_4.

$$
\begin{array}{c} M_4 \\ A \\ B_1 \\ B_2 \\ C \\ D_1 \\ D_2 \\ D_3 \\ E \end{array}
\begin{array}{c} \begin{array}{cccc} m_1 & m_4 & m_2 & m_3 \end{array} \\ \left(\begin{array}{cccc} 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array} \right) \end{array}
$$

Nadaljujemo z iskanjem novih konfliktov. Iz m_2 v grafu H_M ne vodi nobena povezava, zato se pomaknemo na naslednji stolpec v matriki, m_4. Iz vozlišča m_4 v acikličnem digrafu obstaja usmerjena povezava do m_2. Tako kot prej preverimo, ali ima v vsaki vrstici, v kateri ima m_4 vrednost 1, tudi m_2 vrednost 1. Hitro se prepričamo, da to res drži, kar pomeni, da novih konfliktov med m_4 in m_2 nismo ustvarili.

Pomaknemo se na tretji stolpec matrike, m_2. Kot smo že ugotovili, iz vozlišča m_2 v grafu H_M ne vodi nobena povezava, zato se pomaknemo na naslednji in zadnji stolpec,

m_3. Iz njegovega vozlišča m_3 v grafu H_M prav tako ne vodi nobena povezava in s tem je iskanje morebitnih novih konfliktov zaključeno.

$$\begin{array}{c} M_4 \\ A \\ B_1 \\ B_2 \\ C \\ D_1 \\ D_2 \\ D_3 \\ E \end{array} \begin{array}{c} \begin{array}{cccc} m_1 & m_4 & m_2 & m_3 \end{array} \\ \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{array}$$

Po razcepu vseh vrstic in preverjanju ter odpravljanju novih konfliktov, je končni rezultat matrika M_4, ki je brez konfliktov. ▲

4.2.1 Protiprimer pravilnosti delovanja algoritma

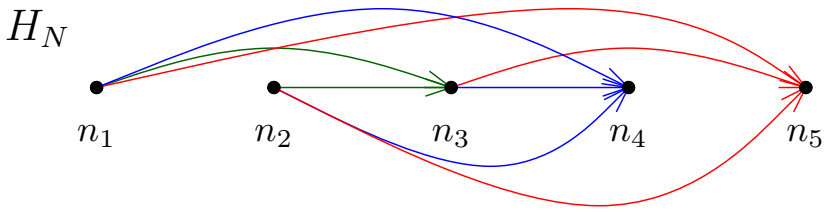
Zgoraj opisani in na primeru predstavljeni algoritem se je skozi reševanje primerov izkazal za napačnega. Utemeljimo to s protiprimerom: opišimo primer vhodnih podatkov, na katerih je končni rezultat izvajanja algoritma konfliktna matrika.

Primer 4.8. Dana je matrika N z dvema konfliktoma: stolpec n_1 je v konfliktu s stolpcem n_2, stolpec n_4 pa s stolpcem n_5.

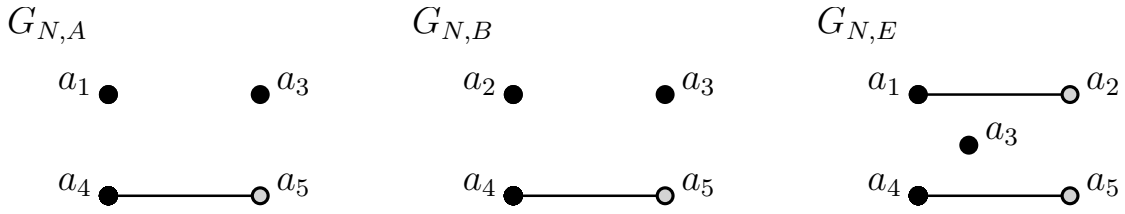
$$\begin{array}{c} N \\ A \\ B \\ C \\ D \\ E \end{array} \begin{array}{c} \begin{array}{ccccc} n_1 & n_2 & n_3 & n_4 & n_5 \end{array} \\ \begin{pmatrix} 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix} \end{array}$$

Graf vsebovanja H_N za matriko N je prikazan na sliki 16. Zaradi večje preglednosti so usmerjene povezave obarvane z različnimi barvami, in sicer so iste barve vse povezave, ki vodijo v isto vozlišče. Vidimo, da stolpec n_5 vsebuje vse stolpce razen stolpca n_4, s katerim je v konfliktu. Podobno velja za stolpec n_4, ki vsebuje vse stolpce razen stolpca n_5, s katerim je v konfliktu. Stolpec n_3 vsebuje n_1 in n_2, slednja pa ne vsebujeta nobenega drugega stolpca.

Stolpci matrike N so že urejeni glede na topološko urejenost grafa H_N, zato permutacija ni potrebna.



Slika 16: Graf vsebovanja H_N



Slika 17: Obarvani grafi G_N konfliktov posameznih vrstic

Obarvani grafi konfliktov vrstic, za katere je potreben razcep, so prikazani na sliki 17. Graf $G_{N,A}$ vrstice A vsebuje 4 vozlišča in 1 povezavo, ki predstavlja konflikt med stolpcema n_4 in n_5. Graf obarvamo z dvema barvama in po razcepu vrstice A dobimo dve novi vrstici, A_1 in A_2. Podobno velja še za vrstico B, katere graf $G_{N,B}$ prav tako vsebuje 4 vozlišča in povezavo, ki predstavlja isti konflikt.

Razcep vrstic C in D ni potreben, saj njuna grafa $G_{N,C}$ in $G_{N,D}$ vsebujeta vsak samo po eno točko. Graf $G_{N,E}$ vrstice E vsebuje 5 točk in povezavi med a_1 in a_2 ter med a_4 in a_5. Obarvamo ga z dvema barvama in E razcepimo na E_1 in E_2.

Po razcepu vrstic A, B in E dobimo sledečo matriko.

$$\begin{array}{c} N_1 \\ A_1 \\ A_2 \\ B_1 \\ B_2 \\ C \\ D \\ E_1 \\ E_2 \end{array} \begin{array}{c} \begin{array}{ccccc} n_1 & n_2 & n_3 & n_4 & n_5 \end{array} \\ \begin{pmatrix} 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \end{pmatrix} \end{array}$$

Po končanem razcepu vrstic se s pomočjo grafa H_N lotimo iskanja in odpravljanja morebitnih na novo nastalih konfliktov. Začnemo v vozlišču n_1, ki predstavlja prvi stolpec matrike. Stolpec ima v matriki vrednosti 1 v vrsticah A_1 in E_1, iz vozlšča n_1 v grafu H_N pa vodijo usmerjene povezave v n_3, n_4 in n_5. Stolpca n_3 in n_4 imata v teh

vrsticah prav tako vrednost 1, medtem ko ima stolpec n_5 v obeh vrednost 0. Novega konflikta torej nismo ustvarili iz, česar sledi tudi, da v matriki, dobljeni po razcepu vrstic, stolpec n_1 ni več vsebovan v stolpcu n_5.

Pomaknemo se na naslednji stolpec, n_2. Iz vozlišča n_2 v H_N prav tako vodijo usmerjene povezave v n_3, n_4 in n_5. Za lažjo predstavljivost so vrednosti stolpcev v vrsticah B_1 in E_2, kjer ima n_2 vrednost 1, prikazane v spodnji podmatriki matrike N_1

$$\begin{array}{c} N_1' \\ B_1 \\ E_2 \end{array} \begin{array}{c} \begin{array}{cccc} n_2 & n_3 & n_4 & n_5 \end{array} \\ \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix} \end{array}$$

Stolpec n_2 ima vrednost 1 vsaj v eni isti vrstici kot vsak stolpec, ki ga vsebuje (n_3, n_4 in n_5), ne pa v vseh. Ustvarili smo torej nove konflikte. S spremembo vrednosti 0 v 1 v vrstici E_2 stolpca n_3 odpravimo nov konflikt med n_2 in n_3, s spremembo vrednosti 0 v 1 v isti vrstici stolpca m_4 pa konflikt med n_2 in n_4. Zadnjega novega konflikta med n_2 in n_5 se znebimo s spremembo 0 v 1 v vrstici B_1 stolpca n_5. Po razreševanju konfliktov dobimo matriko N_2.

$$\begin{array}{c} N_2 \\ A_1 \\ A_2 \\ B_1 \\ B_2 \\ C \\ D \\ E_1 \\ E_2 \end{array} \begin{array}{c} \begin{array}{ccccc} n_1 & n_2 & n_3 & n_4 & n_5 \end{array} \\ \begin{pmatrix} 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 \end{pmatrix} \end{array}$$

Pomaknemo se na naslednji stolpec, n_3, ki ima vrednosti 1 v vrsticah A_1, B_1, E_1 in E_2. Iz vozlišča n_3 v H_N vodita usmerjeni povezavi v n_4 in n_5. V vseh vrsticah, v katerih ima n_3 vrednost 1, ima tudi n_4 vrednosti 1, kar pomeni, da n_3 in n_4 nista v konfliktu. Smo pa ustvarili konflikt med n_3 in n_5, saj ima slednji ima v vrsticah B_1 in E_2 vrednost 1, v A_1 in E_1 pa vrednost 0. Podobno kot prej konflikt razrešimo tako, da vrednosti 0 stolpca n_5 v vrsticah A_1 in E_1 spremenimo v 1. Dobimo matriko N_3.

$$
\begin{array}{c} N_3 \\ A_1 \\ A_2 \\ B_1 \\ B_2 \\ C \\ D \\ E_1 \\ E_2 \end{array}
\begin{array}{c} \begin{array}{ccccc} n_1 & n_2 & n_3 & n_4 & n_5 \end{array} \\ \left(\begin{array}{ccccc} 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \end{array} \right) \end{array}
$$

Pomaknemo se na stolpec n_4. Iz vozlišča n_4 v grafu H_N ne vodi nobena povezava, zato se pomaknemo na naslednji in zadnji stolpec, n_5. Iz njegovega vozlišča n_5 v grafu H_N prav tako ne vodi nobena povezava in s tem je iskanje morebitnih novih konfliktov zaključeno.

Matrika N_3, ki jo dobimo po razcepu vseh vrstic in preverjanju ter opravljanju novih konfliktov, bi morala biti brez konfliktov, vendar opazimo, da sta stolpca n_4 in n_5 še vedno, oziroma ponovno v konfliktu. ▲

V zgornjem primeru 4.8 problem nastane, ko je v grafu $G_{N,E}$ na sliki 17 vozlišču a_2 dodeljena ista barva kot vozlišču a_5, po tem ko je bila v grafu $G_{N,B}$ na isti sliki vozlišču a_2 dodeljena ista barva kot vozlišču a_4, ki je zaradi konflikta med stolpcema n_4 in n_5 povezano z a_5. Posledično dobimo po razcepu vrstice B v vrstici B_1 vrednost 1 v stolpcih n_2 in n_4, po razcepu vrstice E pa v vrstici E_2 vrednost 1 v stolpcih n_2 in n_5. S tem se ustvarita konflikta med stolpcema n_2 in n_4 ter n_2 in n_5. V grafu vsebovanja H_N iz n_2 vodita usmerjeni povezavi tako v n_4 kot v n_5, zaradi česar smo pri odpravljanju teh dveh novih konfliktov v n_4 in n_5 spremenili nekaj vrednosti 0 v 1, da je bil stolpec n_2 ponovno vsebovan v obeh. A ravno s spreminjanjem vrednosti 0 v 1 v stolpcih, ki sta bila v prvotni matriki konfliktna, po razcepu pa smo ju naredili tuja, se konflikt med njima spet pojavi.

Oglejmo si podrobneje, kje in zakaj algoritem zataji. Do problema lahko pride, kadar imamo tri stolpce, od katerih je eden vsebovan v ostalih dveh, ki sta med seboj konfliktna. Recimo da imamo stolpce m_1, m_2 in m_3, v konfliktu sta m_2 in m_3, m_1 pa je vsebovan v obeh, in recimo da podmatrika M' neke večje matrike mutacij M izgleda takole:

$$
\begin{array}{c} M' \\ A \\ B \\ C \\ D \end{array}
\begin{array}{c} \begin{array}{ccc} m_1 & m_2 & m_3 \end{array} \\ \left(\begin{array}{ccc} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{array} \right) \end{array}
$$

Če pri razcepu prve vrstice dodelimo isto barvo stolpcema m_1 in m_2, pri razcepu druge pa m_1 in m_3, je rezultat spodnja matrika.

$$M_1' \begin{array}{c} \\ A_1 \\ A_2 \\ B_1 \\ B_2 \\ C \\ D \end{array} \begin{array}{c} \begin{array}{ccc} m_1 & m_2 & m_3 \end{array} \\ \begin{pmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \end{array}$$

Konflikt med m_2 in m_3 smo sicer odpravili, a z odpravljanjem dveh novih konfliktov med m_1 in m_2 ter m_1 in m_3 s spreminjanjem vrednosti 0 v 1 v vrsticah A_1 v stopcu m_3 in B_1 v stolpcu m_2 se konflikt ponovno pojavi, kot je razvidno iz matrike M_2'.

$$M_2' \begin{array}{c} \\ A_1 \\ A_2 \\ B_1 \\ B_2 \\ C \\ D \end{array} \begin{array}{c} \begin{array}{ccc} m_1 & m_2 & m_3 \end{array} \\ \begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \end{array}$$

Problem je, ker se stolpcu m_1, ki je vsebovan v dveh med seboj konfliktnih stolpcih m_2 in m_3, med dodeljevanjem barv v grafu konfliktov lahko enkrat dodeli ista barva kot m_2, drugič pa ista barva kot m_3. Zagotovo namreč obstaja vsaj ena vrstica, v kateri imajo vsi stolpci vrednost 1, saj sicer m_1 ne bi bil vsebovan v m_2 in m_3. Če obstajata vsaj dve taki vrstici, ima lahko posledično stolpec m_1 po razcepu vrednost 1 v isti vrstici kot m_2 (m_3 ima v tej vrstici zagotovo vrednost 0, saj sta si stolpca m_2 in m_3 po razcepu tuja) in vrednost 1 v isti vrstici kot m_3 (m_2 ima v tej vrstici zagotovo vrednost 0). V takem primeru si para stolpcev m_1 in m_2 ter m_1 in m_3 po razcepu nista tuja, ampak konfliktna. Po pravilih algoritma to odpravimo tako, da se sklicujemo na graf vsebovanja in ohranimo vsa vsebovanja. To pomeni, da morata imeti m_2 in m_3, ki sta si po razcepu tuja, na mestih, kjer ima m_1 vrednost 1, tudi vrednost 1, da bo m_1 vsebovan v njiju. S tem ustvarimo vrstice, v katerih so vrednosti vseh stolpcev enake 1, s čimer odpravimo nova konflikta, a obenem si stolpca m_2 in m_3 zaradi tega nista več tuja, ampak sta ponovno v konfliktu. Vrednosti 0 smo namreč spremenili v 1 ravno v vrsticah, ki so zagotavljale, da sta si bila prvotno konfliktna stolpca m_2 in m_3 po razcepu tuja.

Nemogoče je, da dva med sabo tuja si stolpca oba vsebujeta nek tretji stolpec. Če torej obstaja stolpec, ki je vsebovan v dveh med sabo konfliktnih stolpcih, ki ju z razcepom želimo narediti tuja, mora biti eden od teh konfliktnih stolpcev po razcepu tuj z obema ostalima, sicer se prvotnega konflikta ne rešimo, česar pa algoritem ne upošteva. Na tej manjši podmatriki M' je razvidno, da bi pravilen rezultat dobili že, če bi uporabili drugo barvanje v grafu konfliktov vrstice B, kjer bi namesto m_1 in m_3 isto barvo dodelili vozliščema, ki predstavljata stolpca m_1 in m_2. Na večjih primerih, že recimo na matriki iz prejšnjega primera 4.8, kjer konflikta stolpca n_4 in n_5 oba vsebujeta stolpce n_1, n_2 in n_3, pa je težje opaziti, ali obstaja pravilno barvanje, ki bo matriko razcepilo tako, da bo po končanem izvajanju algoritma brez konfliktov. A tudi če tako barvanje obstaja, ga algoritem ne more uganiti.

Za primere, kjer noben stolpec ni vsebovan v paru konfliktnih stolpcev, je v članku [13] opisan polinomski algoritem, ki problem reši optimalno.

4.2.2 Pravilen algoritem

Namesto grafov konfliktov posameznih vrstic definiramo graf konfliktov G_M za celotno matriko M, v katerem vsako vozlišče ustreza enemu stolpcu, vozlišči pa sta povezani, če sta njuna pripadajoča stolpca v konfliktu. Graf obarvamo, nato pa pri razcepu za vsako vrstico r gledamo podgraf $G_{M,r}$ grafa G_M, ki vsebuje samo tista vozlišča, katerih stolpci imajo v tej vrstici r vrednost 1. Vrstico razcepimo glede na barve vozlišč v $G_{M,r}$, ne ozirajoč se na konflikte. To pomeni, da lahko razcepimo tudi vrstico, katere podgraf $G_{M,r}$ grafa G_M je brez povezav (torej med stolpci matrike ni konfliktov), so pa vozlišča v njem različnih barv. Razcepljena matrika ima tako lahko več vrstic, kot bi bilo mogoče potrebno, saj se hitro pripeti, da sta vozlišči stolpcev, med katerima obstaja relacija vsebovanja, v grafu konfliktov G_M različnih barv, kot bomo kasneje videli tudi na kratkem primeru.

Algoritem je zelo preprost in daje pravilne rezultate, ne pa optimalnih.

Skica dokaza pravilnosti algoritma. Stolpca sta si lahko tuja, konfliktna, ali pa je med njima relacija vsebovanja.

Če sta si stolpca tuja, je vseeno, kakšne barve sta, saj nimata skupnih 1 in jih tudi po razcepu ne moreta imeti, ne glede na izbrano barvanje.

Če sta stolpca konfliktna, sta njuni vozlišči v G_M povezani in posledično različnih barv. Razcepilo se ju bo tako, da si bosta tuja.

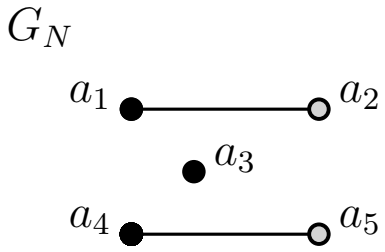
Novi konflikti lahko torej nastanejo samo med stolpci, ki imajo v originalni matriki relacijo vsebovanja. Če je med stolpcema relacija vsebovanja in sta iste barve, potem se to vsebovanje pri razcepu ohrani – kjer imata skupne 1 bodo po definiciji algoritma 1

ostale v istih vrsticah. Če sta različne barve, se bodo vrstice, na katerih imata skupne 1, zaradi različnih barv razcepile tako, da si bosta tuja. □

Graf G_M je lahko poljuben, zato je časovna zahtevnost algoritma eksponentna, oziroma polinomska, če uporabimo polinomsko hevristiko za barvanje grafov.

Sledi primer delovanja algoritma.

Primer 4.9. Vzemimo isto matriko N kot v prejšnjem protiprimeru 4.8 z dvema konfliktoma: med stolpcema n_1 in n_2 ter med n_4 in n_5. Obarvan graf G_N konfliktov celotne matrike je prikazan na sliki 18.



Slika 18: Obarvan graf G_N konfliktov celotne matrike

Podgraf $G_{N,A}$ vrstice A vsebuje štiri vozlišča: a_1, a_3 in a_4, ki so črna in a_5, ki je sivo. Zaradi dveh barv razcepimo A na dve vrstici, A_1 in A_2, pri čemer imajo v prvi novi vrstici vrednost 1 črna vozlišča, v drugi pa sivo. Podobno naredimo z vrsticama B in E.

Rezultat razcepa je matrika N_1, ki je brez konfliktov.

$$
\begin{array}{c} N_1 \\ A_1 \\ A_2 \\ B_1 \\ B_2 \\ C \\ D \\ E_1 \\ E_2 \end{array}
\begin{array}{c}
\begin{array}{ccccc} n_1 & n_2 & n_3 & n_4 & n_5 \end{array} \\
\left(\begin{array}{ccccc}
1 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 & 0 \\
0 & 1 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 \\
1 & 0 & 1 & 1 & 0 \\
0 & 1 & 0 & 0 & 1
\end{array}\right)
\end{array}
$$

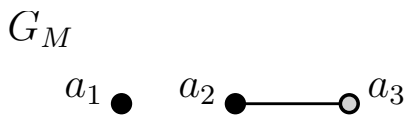
▲

V zgornjem primeru 4.9 lahko opazimo, da si je stolpec n_3 v končni matriki tuj s stolpcem n_5, ki ga je v prvotni matriki vseboval. Zaradi različnih barv vozlišč a_3 in a_5 sta stolpca skozi razcep v vseh vrsticah, kjer sta imela prej skupne vrednosti 1, prejela 1 v različnih novih vrsticah. S tem smo tudi rešili problem prvotnega algoritma, ki je

predstavljen v primeru 4.8. Opazimo lahko še, da je končno število vrstic razcepljene matrike pri obeh algoritmih enako, kar pa v splošnem ne velja nujno. Na zelo enostavnem primeru si oglejmo, kdaj in zakaj je število novih vrstic pravilnega algoritma lahko večje od spodnje meje, podane v lemi 4.5.

Primer 4.10. Dana je matrika M s tremi vrsticami in tremi stolpci. V konfliktu sta stolpca m_2 in m_3, graf konfliktov G_M je na sliki 19.

$$\begin{array}{c} M \\ A \\ B \\ C \end{array} \begin{array}{c} \begin{array}{ccc} m_1 & m_2 & m_3 \end{array} \\ \begin{pmatrix} 0 & 1 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} \end{array}$$



Slika 19: Obarvan graf G_M konfliktov celotne matrike

Razcepiti moramo vrstici A, katere podgraf grafa konfliktov G_M vsebuje vozlišči konfliktnih stolpcev a_2 in a_3, in C, kjer sta v podgrafu vozlišči a_1 in a_3, med katerima sicer ni konflikta, sta pa različne barve. Razcepljena matrika M_1 je brez konfliktov.

$$\begin{array}{c} M_1 \\ A_1 \\ A_2 \\ B \\ C_1 \\ C_2 \end{array} \begin{array}{c} \begin{array}{ccc} m_1 & m_2 & m_3 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \end{array}$$

▲

Algoritem iz članka vrstice C ne bi razcepljal, saj med vozlišči njunih stolpcev v grafu konfliktov ni povezave, naš algoritem pa se na to ne ozira, ampak razceplja izključno samo glede na barve vozlišč. Posledično je končno število vrstic razcepljene matrike lahko večje od spodnje meje, podane v članku, ki je enaka vsoti kromatičnih števil grafov konfliktov posamezne vrstice.

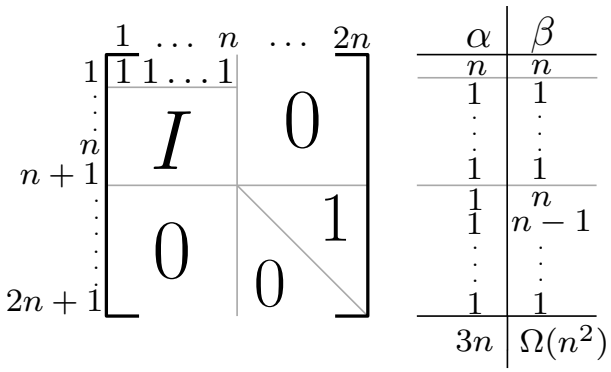
Na tem mestu nas seveda zanima, ali je naš pravilni algoritem primerljiv s spodnjo mejo, podano v lemi 4.5, ob predpostavki, da graf G_M optimalno obarvamo. Označimo to spodnjo mejo z $\alpha(M) = \sum_r \chi(G_{M,r})$. Naj bo $\beta(M) = \max\{\sum_r |c(V(G_{M,r}))| : c$ je optimalno barvanje grafa $G_M\}$. V definiciji parametra $\beta(M)$ smo upoštevali najslabši

možen primer, saj algoritem ne razlikuje med različnimi optimalnimi barvanji grafa G_M. Očitno je torej $\beta(M)$ zgornja meja za število vrstic matrike, ki jo izračuna algoritem.

Žal se izkaže, da je razmerje med številom vrstic v rešitvi, ki jo izračuna algoritem, in spodnjo mejo iz leme 4.5 neomejeno, in to za družino matrik, za katere je optimalna vrednost problema dosežena s spodnjo mejo. Algoritem torej ni c-aproksimacijski za noben $c \geq 1$.

Primer matrik, na katerih algoritem to zgornjo mejo lahko doseže, je podan na sliki 20. Glede na rezultat iz članka [13] za te matrike velja tudi, da je vrednost optimalne rešitve enaka $\alpha(M)$.

Matrika ima $2n + 1$ vrstic in $2n$ stolpcev, od katerih je prvih n med seboj paroma konfliktnih, med preostalimi pari pa veljajo relacije vsebovanja. Razvidno je, da algoritem prvo polovico vrstic razcepi optimalno. Za prvo vrstico, katere graf konfliktov je poln graf, uporabi n barv. Prvi vrstici sledi *identična* podmatrika (matrika, ki ima po diagonali vrednost 1, drugod pa 0), katere grafi konfliktov vsebujejo samo po eno vozlišče in so obarvljivi z eno barvo. Problem se pojavi v drugi polovici, kjer je prisotna *zgornje trikotna* podmatrika, torej matrika, v kateri so pod diagonalo vrednosti 0, po diagonali in nad njo pa vrednosti 1. To pomeni, da vsak naslednji stolpec vsebuje prejšnjega in med njimi ni nobenega konflikta. V optimalni rešitvi razcep teh vrstic ni potreben, saj so njihovi grafi konfliktov brez povezav in torej obarvljivi z eno barvo. Naš algoritem pa je lahko pri barvanju celotnega grafa konfliktov tem n stolpcev dodelil n različnih barv. V tem primeru prvo vrstico te zgornje trikotne podmatrike razcepi na n vrstic, drugo na $n - 1$ vrstic in tako dalje do zadnje vrstice, ki vsebuje vrednost 1 samo v zadnjem stolpcu. Vidimo lahko, da za zgornjo mejo velja $\beta(M) = \Omega(n^2)$.



Slika 20: Zgornja meja $\beta(M)$ pravilnega algoritma

Spodnja meja števila vrstic matrike, ki jo izračuna algoritem, zaenkrat v splošnem ostaja neznana.

5 Zaključek

V zaključni nalogi smo spoznali popolno filogenijo, njeno povezanost s tetivnimi grafi in barvanjem grafov ter eno njenih novejših razširitev, mešano popolno filogenijo. Videli smo, kako se problem gradnje popolnega filogenetskega drevesa iz vhodnih podatkov prevede na problem iskanja c-triangulacije particijskega presečnega grafa na teh vhodnih podatkih. Na primerih smo si ogledali tudi algoritem za preverjanje tetivnosti grafa z odstranjevanjem klik ter konstrukcijo popolnega filogenetskega drevesa tako iz tetivnega grafa kot iz binarne matrike.

Velika pozornost je bila namenjena problemu mešane popolne filogenije, za katerega veliko vprašanj ostaja odprtih. Predstavljena in podrobno opisana je nepravilnost Hajirasoulihovega in Raphaelovega algoritma [11] za optimalno rešitev problema. Predlagani algoritem, ki je formuliran na koncu zadnjega poglavja, problem sicer reši pravilno, a v splošnem ne daje optimalnih rezultatov.

Pri nadaljnjem raziskovanju bi bil torej zanimiv razvoj pravilnega algoritma, ki bi problem mešane popolne filogenije rešil optimalno. Za boljše razumevanje problemov in nadaljno branje na to temo je priporočljiv članek Finding a perfect phylogeny from mixed tumor samples [13], ki sicer zaenkrat še ni objavljen, obravnava pa nekatere nepravilnosti in zahtevnost problema minimalnega razcepa vrstic, ki je bistven za rešitev problema mešane popolne filogenije.

6 Literatura

[1] R. Agarwala in D. Fernández-Baca, A polynomial-time algorithm for the perfect phylogeny problem when the number of character states is fixed, *SIAM Journal on Computing* 23 (1994), 1216–1224. *(Citirano na strani 16.)*

[2] D. Fernández-Baca, The perfect phylogeny problem, v: X. Cheng in D. Z. Du (ur.), *Steiner Trees in Industry*, Kluwer Academic Publishers, 2000, 1–32. *(Citirano na straneh 9, 10 in 17.)*

[3] H. L. Bodlaender, M. R. Fellows in T. J. Warnow, Two strikes against perfect phylogeny. V *Automata, Languages and Programming: Proc. of the 19th International Colloquium*, Springer-Verlag, 1992, 273–283. *(Citirano na straneh 9 in 14.)*

[4] H. Bodlaender in T. Kloks, A simple linear time algorithm for triangulating three-colored graphs, *Journal of Algorithms* 15 (1993), 160–172. *(Citirano na strani 16.)*

[5] D. Brélaz, New methods to color the vertices of a graph, *Communications of the ACM* 22 (1979), 251–256. *(Citirano na straneh 2 in 8.)*

[6] P. Buneman, A characterisation of rigid circuit graphs, *Discrete Mathematics* 9 (1974), 205–212. *(Citirano na straneh 1, 4, 5 in 10.)*

[7] M. R. Garey, D. S. Johnson in L. Stockmeyer, Some simplified NP-complete problems. V *Proceedings of the Sixth Annual ACM Symposium on Theory of Computing*, 1974, 47–63. *(Citirano na strani 8.)*

[8] F. Gavril, The intersection graphs of subtrees in trees are exactly the chordal graphs, *Journal of Combinatorial Theory, Series B* 16 (1974), 47–56. *(Citirano na straneh 1, 4 in 5.)*

[9] M. C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs (Annals of Discrete Mathematics, vol. 57), Second Edition*, North-Holland Publishing, 2004. *(Citirano na straneh 4 in 5.)*

[10] D. Gusfield, Efficient algorithms for inferring evolutionary trees, *Networks* 21 (1991), 19–28. *(Citirano na straneh 1, 16, 17 in 21.)*

[11] I. Hajirasouliha in B. J. Raphael Reconstructing mutational history in multiply sampled tumors using perfect phylogeny mixtures. V *Algorithms in Bioinformatics - 14th International Workshop, WABI 2014, Wroclaw, Poland*, 2014, 354–367. *(Citirano na straneh 1, 23, 25 in 42.)*

[12] A. Hertz in D. de Werra, Using Tabu Search Techniques for Graph Coloring, *Computing* 39 (1987), 345–351. *(Citirano na straneh 2 in 8.)*

[13] A. Hujdurović, U. Kačar, M. Milanič, B. Ries in A. I. Tomescu, *Finding a perfect phylogeny from mixed tumor samples*, Poslano v objavo, 2015. *(Citirano na straneh 2, 27, 28, 38, 41 in 42.)*

[14] S. Kannan in T. J. Warnow, Triangulating three-colored graphs, *SIAM Journal on Discrete Mathematics* 5 (1992), 249–258. *(Citirano na strani 16.)*

[15] S. Kannan in T. J. Warnow, Inferring evolutionary history from DNA sequences, *SIAM Journal on Computing* 23 (1994), 713–737. *(Citirano na strani 16.)*

[16] S. Kannan in T. J. Warnow, A fast algorithm for the computation and enumeration of perfect phylogenies, *SIAM Journal on Computing* 26 (1997), 1749–1763. *(Citirano na strani 16.)*

[17] A. Lim, Y. Zhu, Q. Lou in B. Rodrigues, Heuristic Methods for Graph Coloring Problems. V *Proceedings of the 2005 ACM Symposium on Applied Computing*, 2005, 933–939. *(Citirano na strani 8.)*

[18] F. R. McMorris, T. J. Warnow in T. Wimer, Triangulating vertex-colored graphs, *SIAM Journal on Discrete Mathematics* 7 (1994), 296–306. *(Citirano na strani 16.)*

[19] M. A. Steel, The complexity of reconstructing trees from qualitative characters and subtrees, *Journal of Classification* 9 (1992), 91–116. *(Citirano na strani 14.)*

[20] J. R. Walter, Representations of chordal graphs as subtrees of a tree, *Journal of Graph Theory* 2 (1978), 265–267. *(Citirano na straneh 1 in 4.)*

[21] T. J. Warnow, *Some combinatorial optimization problems in phylogenetics*, University of Pennsylvania, 1999. *(Citirano na straneh 9, 10, 15 in 16.)*

[22] T. J. Warnow, *Combinatorial Algorithms for Constructing Phylogenetic Trees*, Ph.D. Thesis, University of California, Berkeley, CA, 1991. *(Citirano na strani 15.)*

[23] D. B. West, *Introduction to Graph Theory, Second Edition*, Prentice Hall, 2001. *(Citirano na straneh 5 in 8.)*