Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Urban Soban

Implementacija metode asimetričnega srednjega drevesa za iskanje konsenza filogenetskih dreves

DIPLOMSKO DELO

UNIVERZITETNI ŠTUDIJSKI PROGRAM PRVE STOPNJE
RAČUNALNIŠTVO IN INFORMATIKA

Mentor: doc. dr. Tomaž Curk

Ljubljana 2014

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

[1]V dogovoru z mentorjem lahko kandidat diplomsko delo s pripadajočo izvorno kodo izda tudi pod katero izmed alternativnih licenc, ki ponuja določen del pravic vsem: npr. Creative Commons, GNU GPL. V tem primeru na to mesto vstavite opis licence, na primer tekst [?].

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Besedilo teme diplomskega dela študent prepiše iz študijskega informacijskega sistema, kamor ga je vnesel mentor. V nekaj stavkih bo opisal, kaj pričakuje od kandidatovega diplomskega dela. Kaj so cilji, kakšne metode uporabiti, morda bo zapisal tudi ključno literaturo.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Urban Soban, z vpisno številko **63100344**, sem avtor diplomskega dela z naslovom:

Implementacija metode asimetričnega srednjega drevesa za iskanje konsenza filogenetskih dreves

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Tomaža Curka
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 11. januarja 2011 Podpis avtorja:

Na tem mestu zapišite, komu se zahvaljujete za izdelavo diplomske naloge. Pazite, da ne boste koga pozabili. Utegnil vam bo zameriti. Temu se da izogniti tako, da pozabite na celo zahvalo.

Družini.

Kazalo

Seznam uporabljenih kratic

kratica	**angleško**	**slovensko**
AMT	asymmetric median tree	asimetrično srednje drevo
MIS	maximum independent set	največja neodvisna množica
UPGMA	unweighted pair group method with arithmetic mean	neuteženo gručanje s pomočjo aritmetične sredine
CDAO	comparative data analysis ontology	ontologija podatkov za primerjalno analizo

Povzetek

V vzorcu je predstavljen postopek priprave diplomskega dela z uporabo okolja LaTeX. Vaš povzetek mora sicer vsebovati približno 100 besed, ta tukaj je odločno prekratek.

Ključne besede: konsenz, drevo, filogenetika.

Abstract

This sample document presents an approach to typesetting your BSc thesis using LaTeX. A proper abstract should contain around 100 words which makes this one way too short.

Keywords: consensus, tree, phylogeny.

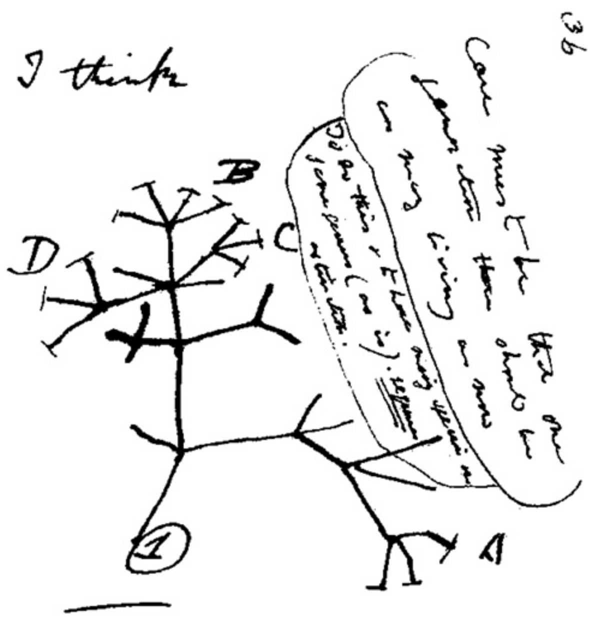
Poglavje 1

Uvod

O evolucijskih razmerjih med živimi organizmi se je prvi spraševal Charles Darwin, ko je narisal znamenito "drevo življenja," prikazano na sliki 1.1. Vendar je Darwin lahko organizme razvrščal le glede na njihove morfološke lastnosti. Nato je prišlo do odkritja DNA in izuma računalnika ter porodila se je ideja o računski filogenetiki, eni izmed prvih področij bioinformatike. Cilj računske filogenetike je odkriti evolucijska razmerja med različnimi taksonomskimi enotami na podlagi sekvenc DNA in RNA. Mnoge metode računske filogenetike so bile razvite že v 1970-ih, vendar so nekatere prišle v praktično uporabo šele v zadnjem času, ko so računalniki postali dovolj zmogljivi. Ker različne metode ali pa celo ena sama metoda lahko proizvede več različnih filogenetskih dreves, mnogokrat želimo rezultate kombinirati v eno drevo in tako pridobiti eno teorijo o evolucijski zgodovini taksonomskih enot.

Na pomoč priskočijo konsenzne metode, ki na podlagi različnih kriterijev dele vhodnih dreves v končnem sestavljenem drevesu kombinirajo, ohranijo ali zavržejo, in sicer s ciljem sestaviti drevo, ki kar se da dobro povzema informacije o evolucijski zgodovini, ki jih nosijo vhodna drevesa. Med konstrukcijo konsenznega drevesa lahko metoda izračuna več različnih dreves, izbrano pa je tisto z največjo podporo vhodnih dreves.

V prvem delu diplomske naloge se bomo na kratko seznanili z najbolj uporabljenimi metodami računanja filogenetskih dreves, katerih produkt je



Slika 1.1: "Drevo življenja", prva skica evolucijske zgodovine, narisal jo je Charles Darwin leta 1837 [1].

nato vhod v algoritem za izračun konsenza. Nato bomo obravnavali nekatere priljubljene konsenzne metode, kot so striktni konsenz, večinski konsenz in srednji konsenz (konsenz mediane). V drugem delu naloge se bomo osredotočili na novejšo konsenzno metodo, tj. algoritem asimetričnega srednjega drevesa.

Spoznali bomo teoretično ozadje konstrukcije asimetričnega srednjega drevesa, nato pa bo sledila predstavitev programskega paketa Biopython, v katerega smo vključili implementacijo asimetričnega srednjega drevesa, in ostalih orodij, ki smo jih pri tem uporabili. Uspešnost implementirane metode bomo eksperimentalno ocenili na treh vhodnih množicah, ki izražajo različne lastnosti, in na eni realni vhodni množici. Rezultate bomo primerjali s tremi popularnimi konsenznimi metodami glede na razrešenost končnega drevesa in glede na Robinson-Fouldsovo metriko. Za konec bomo preverili, za kakšno število dreves v vhodni množici je uporaba implementirane metode še dovolj hitra, in predlagali možne izboljšave.

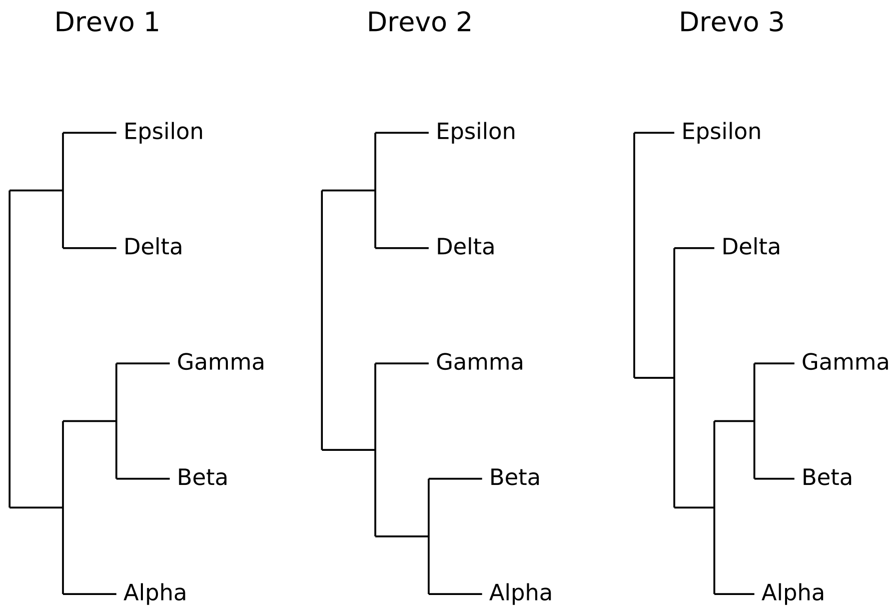
Poglavje 2

Filogenetske in konsenzne metode

V diplomski nalogi se povečamo konsenzni metodi asimetričnega srednjega drevesa. Da bi razumeli, zakaj so konsenzne metode v računski filogenetiki potrebne, bomo v tem razdelku zgolj na kratko predstavili, kako so filogenetska drevesa sploh zgrajena in kakšno vlogo pri tem igrajo konsenzne metode.

2.1 Filogenetske računske metode

Filogenetske računske metode lahko razdelimo na dve večji skupini in sicer distančne in statistične metode. Vse kot vhod prejmejo DNA- ali RNA-sekvence taksonomskih enot, katerih evolucijsko zgodovino želimo rekonstruirati, kot izhod pa vrnejo eno ali več filogenetskih dreves, ki imajo konice (liste) označene z imeni taksonomskih enot. Nekatere metode so zmožne oceniti tudi dolžino vej drevesa, pri čemer dolžina veje predstavlja čas, ki je bil potreben za divergenco dveh taksonomskih enot iz skupnega prednika. Izračun dolžin vej sicer ni odvisen zgolj od izbrane metode, temveč tudi od izbranega modela molekularne ure. Primere treh filogenetskih dreves za pet taksonomskih enot prikazuje slika 2.1. Drevesa na sliki imajo dolžine vseh vej enake, med seboj pa se razlikujejo po topologiji.



Slika 2.1: Primer treh filogenetskih dreves z različnimi topologijami.

2.1.1 Distančne metode

Distančne metode iz vhodnih sekvenc DNA ali RNA s pomočjo izbranega evolucijskega modela najprej generirajo distančno matriko, v kateri so zapisane razdalje med vsemi pari sekvenc. Razdalje predstavljajo divergentne čase med dvema taksonomskima enotama, zato evolucijske modele lahko uporabimo tudi v kombinaciji z drugimi metodami za izračun dolžin vej filogenetskega drevesa. Za generiranje distančne matrike imamo na voljo več različnih evolucijskih modelov, med njimi:

- Jukes-Cantor (JC69) je najbolj enostaven model, ki predpostavlja, da je frekvenca vseh nukleotidov enaka in da so vse možne substitucije na enem baznem paru enako verjetne [9];
- Kimura (K80) model, ki substitucije baznega para razlikuje glede na tip in upošteva, da se lahko določen tip substitucije pojavi bolj pogosto [10]. Tipe substitucij delimo na tranzicije (sprememba purina v purin ali sprememba pirimidina v pirimidin) ter transverzije (sprememba purina

v pirimidina ali obratno) [5];

- General Time Reversible (GTR) je najnaprednejši model, ki evolucijo modelira kot stohastični proces. Ne predpostavlja neodvisnosti vseh lokacij vhodnih sekvenc, temveč šteje frekvence pojavitve nukleotidov glede na pozicijo v kodonu. Nato oceni šest parametrov, s pomočjo katerih zgradi matriko z verjetnostmi prehoda enega nukleotida v drugega [13].

Na podlagi razdalj, zapisanih v distančni matriki, izbrana metoda v gručo uvrsti po dve najbolj podobni taksonomski enoti hkrati (oz. povedano drugače, najbolj podobnima taksonomskima enotama določi skupnega prednika), dokler v gruče ne uvrsti vseh taksonomskih enot. Primera algoritmov, ki spadata v razred distančnih metod, sta UPGMA in metoda združevanja sosedov (angl., neighbor joining).

2.1.2 Metoda največje varčnosti

Metoda največje varčnosti sodi med starejše, vendar še vedno zelo uporabljane metode. Glavna ideja algoritma je analogna principu Occamovega rezila - med več med seboj tekmujočimi hipotezami evolucije izberemo tisto, ki minimizira število evolucijskih dogodkov. Metoda pregleda vsa potencialna drevesa in izbere tisto, ki vhodne sekvence lahko pojasni z najmanjšim številom potrebnih substitucij baznih parov. Tako drevo imenujemo najbolj varčno drevo (angl., most parsimonious tree) [14]. Najbolj varčnih dreves je lahko več, zato je izhodna množica najbolj varčnih dreves idealen primer vhodne množice za eno izmed metod za grajenje konsenznega drevesa.

2.1.3 Metoda največjega verjetja

Metoda maksimalnega verjetja (angl., maximum likelihood) je ena izmed statističnih metod za grajenje filogenetskih dreves. Če za neko drevo predpostavimo, da predstavlja pravilno evolucijsko zgodovino, potem njegovo verjetje predstavlja verjetnost pojavitve vhodnih sekvenc pri danem drevesu.

Metoda poišče drevo, ki verjetje maksimizira in ob tem predpostavlja, da so verjetnosti substitucij na različnih baznih parih medsebojno neodvisne. Najprej izračuna, kako verjetno je predlagano drevo za vsako lokacijo vhodnih sekvenc posebej, nato pa še produkt teh vrednosti, ki predstavlja končno verjetje drevesa. Verjetje drevesa ne predstavlja verjetnosti, da je to drevo dejansko pravilno.

2.1.4 Bayesova inferenca

Bayesova inferenca je še en statistični pristop k iskanju filogenetskega drevesa. Zamisli o uporabi Bayesovega teorema so se pojavile že konec šestdesetih let, vendar do nedavnega implementacija takih algoritmov ni bila smiselna zaradi velike računske zahtevnosti. Od metode maksimalnega verjetja se razlikuje zaradi uporabe vnaprej izbrane apriorne porazdelitve verjetnosti dreves [5]. Stroka ima zaradi tega sicer deljena mnenja o primernosti uporabe, vendar s tem pridobimo lepo lastnost, in sicer zmožnost interpretacije rezultata kot porazdelitev verjetnosti dreves glede na vhodne sekvence.

2.1.5 Vzorčenje primerov

Ker nobena izmed metod ne zagotavlja pravilno generirane evolucijske zgodovine oz. pravilnosti ne moremo preveriti (razen v primeru laboratorijskih organizmov, kjer je razmerje vnaprej znano) [23], raziskovalci po navadi uporabijo eno izmed metod vzorčenja primerov, npr. metoda izloči enega (angl., jackknife) ali metoda razmnoževanja primerov (angl., bootstrap), s katero vzorčijo lokacije vhodnih sekvenc, in za vsak vzorec zgradijo svoje filogenetsko drevo. Vsako drevo, zgrajeno iz vzorca, se primerja s prvotno pridobljenim in izračuna se delež dreves, ki vsebujejo veje prvotnega drevesa. S pomočjo teh deležev lahko ocenimo, kolikšna je negotovost prvotno izračunane topologije filogenetskega drevesa [5].

Težava metod ponovnega vzorčenja je, da ocenjujejo zgolj negotovost v okviru ene metode. Raziskovalci lahko pridobijo več nasprotujočih si hipotez o evolucijski zgodovini, naj si bo z uporabo različnih metod za računanje filogenetskega drevesa ali iz različnih virov podatkov. Ob tem se pojavi potreba po eni hipotezi o evoluciji, ki bi kar se da dobro zajemala vse evolucijske dogodke, ki jih nosijo med seboj tekmujoče si hipoteze. To lahko dosežemo z uporabo konsenznih metod.

2.2 Konsenzne metode

Konsenzne metode nad filogenetskimi drevesi so uporabljene po vsakem filogenetskem algoritmu, ki kot izhod proizvede več filogenetskih dreves. Načeloma lahko delujejo nad katero koli množico dreves, v kateri imajo drevesa enake množice oznak listov. V tem se bistveno razlikujejo od metod superdreves, ki so sposobne kombinirati tudi drevesa, katerih množice oznak listov niso enake [7]. Poleg metode asimetričnega srednjega drevesa, ki jo obravnavamo v nadaljevanju tega dela, naštejmo nekatere najbolj pogoste konsenzne metode:

- Kompatibilno drevo je sestavljeno iz delov vseh vhodnih dreves; ni nujno, da za vhodno množico dreves kompatibilno drevo dejansko tudi obstaja [2].

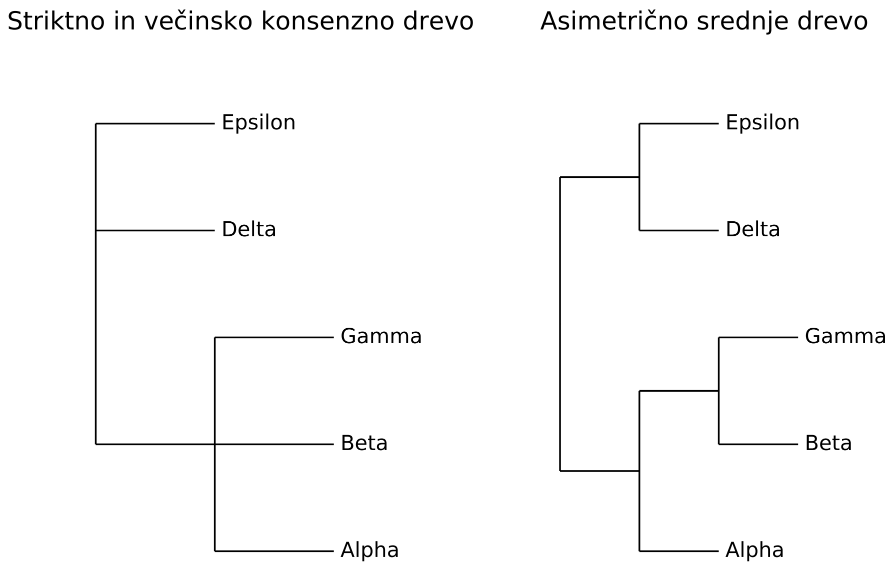
- Striktni konsenz je najbolj konservativna metoda, saj v končnem drevesu ohrani le tiste dele dreves, ki so prisotni v vseh drevesih vhodne množice [7]. Primer striktnega konsenznega drevesa za vhodna drevesa s slike 2.1 je prikazan na sliki 2.2.

- Večinski konsenz vsebuje le tiste dele drevesa, ki so prisotni v več kot polovici dreves vhodne množice [7]. Primer večinskega konsenznega drevesa za vhodna drevesa s slike 2.1 je prikazan na sliki 2.2.

- Srednji konsenz oz. mediana je drevo, ki minimizira vsoto simetričnih razlik glede na drevesa v vhodni množici. Večinsko drevo je prav tako srednje drevo, kar pomeni, da vedno obstaja vsaj eno srednje drevo [2].

- Adamov konsenz je drevo, ki ohrani strukturo poddreves, prisotnih v vseh vhodnih drevesih. Vedno obstaja, vendar lahko vsebuje poddrevesa, ki niso prisotna v nobenem drevesu vhodne množice [3].

- Nelson-Pageva konsenza metoda je bila osnovana na ideji, da se eno drevo lahko moti, dve drevesi pa ne. Predpostavlja, da so deli dreves, prisotni v dveh ali več vhodnih drevesih, zelo verjetno resnični. Take dele imenujemo replicirane komponente. Nelson-Pagevo drevo je tako drevo, ki vsebuje vse replicirane komponente vhodne množice in vse preostale dele dreves vhodne množice, ki so z repliciranimi komponentami kompatibilni [3].



Slika 2.2: Na levi strani je prikazano striktno in večinsko konsenzno drevo za vhodna drevesa s slike 2.1. Drevesi sta v tem primeru enaki. Na desni je za isto vhodno množico prikazano asimetrično srednje drevo.

Večina konsenznih metod ignorira dolžine vej v vhodnih drevesih, tudi če so te navedene. Namen konsenznih metod je torej pridobiti topologijo drevesa, s katero se čim bolj strinjajo vhodna drevesa, ne pa tudi usklajevanje divergentnih časov taksonomskih enot in njihovih skupnih prednikov.

2.3 Programska oprema za izračun filogenetskih dreves

Vse zgoraj opisane metode so že implementirane v samostojnih programskih paketih. Med najbolj znane sodijo:

- ClustalW2 Phylogeny, ki omogoča izračun filogenetskega drevesa s pomočjo distančnih metod UPGMA in združevanja sosedov, korekcijo razdalj pa lahko opravi s pomočjo evolucijskega modela K80 [11]. Poleg možnosti samostojne namestitve ga lahko uporabljamo tudi preko spletnega vmesnika, dostopnega na `http://www.ebi.ac.uk/Tools/phylogeny/clustalw2_phylogeny/`.

- PHYLIP, paket samostojnih programov, ki ponuja izračun filogenetskih dreves s pomočjo distančnih metod, metode največje varčnosti in metode največjega verjetja. Poleg tega ponuja programe za vzorčenje primerov in program za izračun konsenznega drevesa s pomočjo striktnega ter večinskega konsenza. Korekcije razdalj je mogoče opraviti z evolucijskimi modeli JC69, K80 in F84 [12].

- MrBayes, ki izračun filogenetskega drevesa opravi s pomočjo bayesove inference. Za izračun posteriorne verjetnostne porazdelitve uporablja markove verige v kombinaciji z Metropolis-Hastingsovim algoritmom za učinkovito preiskovanje prostora topologij. Ker na izhodu proizvede mnogo dreves, ponuja tudi izračun konsenznega drevesa s pomočjo metode večinskega konsenza [15].

- MEGA6, eno najbolj celovitih programskih okolij za izračun filogenetskih dreves, med drugim ponuja poravnavanje sekvenc pred njihovo uporabo z distančnimi metodami, metodami največje varčnosti ali metode maksimalnega verjetja. Ponuja široko paleto evolucijskih modelov z izjemo modela GTR, izračun konsenznih dreves s pomočjo striktnega ali večinskega konsenza in možnost ponovnega vzorčenja. Program

MEGA6 lahko uporabljamo preko ukazne vrstice, vse funkcionalnosti pa so podprte tudi preko grafičnega vmesnika [16].

- HashCS, program za izračun konsenza filogenetskih dreves, ki šteje za enega najhitrejših programov te vrste. Implementira lastni algoritem, ki teče v polinomskem času. Ker je bil izdelan z namenom hitrejšega povzemanja dreves, ki so rezultat enega izmed algoritmov z uporabo bayesove inference, predpostavlja, da je število vhodnih dreves mnogo večje kot število taksonomskih enot [19].

Poglavje 3

Asimetrično srednje drevo

Metoda asimetričnega srednjega drevesa je nastala ob opažanju, da navadno srednje drevo kaznuje tiste dele dreves, ki ne nastopajo vsaj v polovici dreves vhodne množice, čeprav bi bilo smotrno, da se uporabi kar se da veliko informacije iz vseh vhodnih dreves [2].

Metoda kot vhod prejme množico k filogenetskih dreves $T = \{T_1, T_2, ..., T_k\}$, ki imajo liste označene z n taksonomskimi enotami iz množice $S = \{s_1, s_2, ..., s_n\}$. Sledijo naslednji koraki algoritma, katerega dele bomo podrobneje obravnavali v naslednjih poglavjih:

- drevesa pretvorimo v primerno reprezentacijo,
- poiščemo nekompatibilne pare poddreves in konstruiramo graf nekompatibilnosti,
- v grafu nekompatibilnosti poiščemo največjo neodvisno množico vozlišč in
- rekonstruiramo drevo iz vozlišč največje neodvisne množice.

3.1 Kodiranje dreves

Izbira predstavitve drevesa je pomembna za njihovo nadaljnjo manipulacijo. Izhajamo iz opažanja, da vsaka povezava v drevesu (rob), $e \in E(T_i)$, drevo razdeli na dva dela oz. particiji. Do vsakega lista (taksonomske enote s) vodi enolična pot, predstavljena z množico robov, ki na tej poti nastopajo. Označimo jo s $\pi(s)$. Nato definiramo preslikavo dveh argumentov (3.1), ki zavzame vrednost 1, če pot do podanega lista poteka preko podanega roba, sicer pa 0. $c(e, s)$ za vsak rob $e \in E(T_i)$ torej definira biparticijo drevesa T_i [2]. V prvi množici so listi, do katerih lahko pridemo, če pot vodi preko roba e, v drugi pa tisti, do katerih v tem primeru ne moremo.

$$c(e, s) = \begin{cases} 1 & e \in \pi(s) \\ 0 & e \notin \pi(s) \end{cases} \tag{3.1}$$

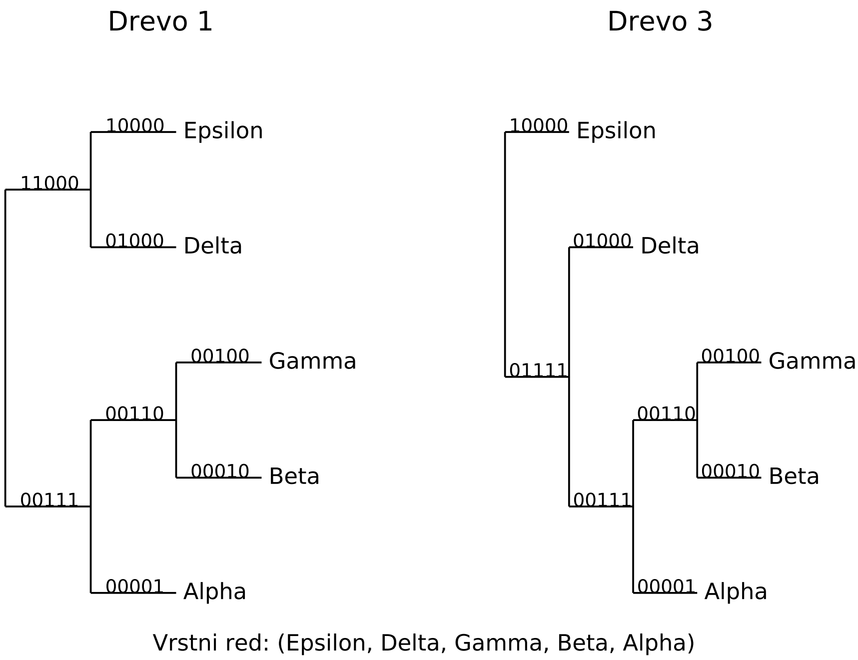
Vsak rob drevesa določa poddrevo, to pa predstavlja skupnega prednika s svojimi potomci. Takemu poddrevesu pravimo tudi klad (taksonomsa skupina, angl., clade). Ne bo nas zanimala celotna struktura poddreves, temveč zgolj njihovi listi. Vsak klad drevesa lahko predstavimo z binarnim nizom, ki ga dobimo z uporabo preslikave 3.2, pri čemer je treba poudariti, da je vrstni red posameznih znakov v binarnem nizu pomemben, saj je na vsak znak vezan pomen; vsako posamezno mesto v nizu pripada enemu izmed listov drevesa, znak na tem mestu pa zgolj indicira prisotnost ali neprisotnost lista oz. taksonomske enote v kladu, kodiranem z binarnim nizom.

$$c_e = \{c(e, s) : s \in S\} \tag{3.2}$$

Če binarne nize vseh kladov drevesa zberemo v množico (3.3), nam $C(T_i)$ določa kodiranje celotnega drevesa T_i. Primer dveh takih množic je prikazan na sliki 3.1, kjer so binarni nizi prikazani na vejah, ki vodijo do korenov kladov.

$$C(T_i) = \{c_e : e \in E(T_i)\} \tag{3.3}$$

Kodiranje zdaj lahko izvedemo nad vsemi drevesi vhodne množice T, pri čemer je ponovno treba poudariti, da se pomen posameznih mest v binarnih



Slika 3.1: Vsi binarni nizi za prvo in tretje drevo s slike 2.1.

nizih med različnimi drevesi ne sme spreminjati, saj sicer kladov iz različnih dreves ne bi mogli medsebojno primerjati.

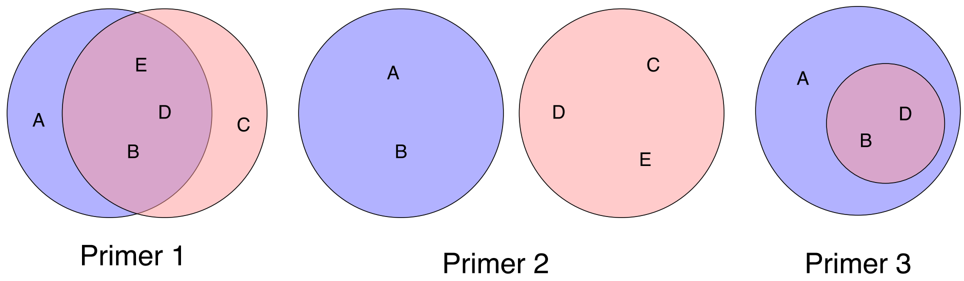
3.2 Kompatibilnost binarnih nizov

Za konstrukcijo grafa nekompatibilnosti najprej potrebujemo kriterij kompatibilnosti kladov oz. binarnih nizov, ki jih kodirajo. Za potrebe lažje predstavitve kompatibilnosti za vsak binarni niz c_e definiramo množico $\phi(c_e)$ (3.4), ki vsebuje oznake taksonomskih enot, prisotnih v kladu oz. njegovem pripadajočem binarnem nizu c_e.

$$\phi(c_e) = c_e^{-1}(1) = \{s \in S : c(e, s) = 1\} \tag{3.4}$$

$$\phi(j) \cap \phi(k) = \emptyset \vee \phi(j) \subseteq \phi(k) \vee \phi(k) \subseteq \phi(j) \tag{3.5}$$

Če za binarna niza j in k iz poljubnih dveh dreves vhodne množice T



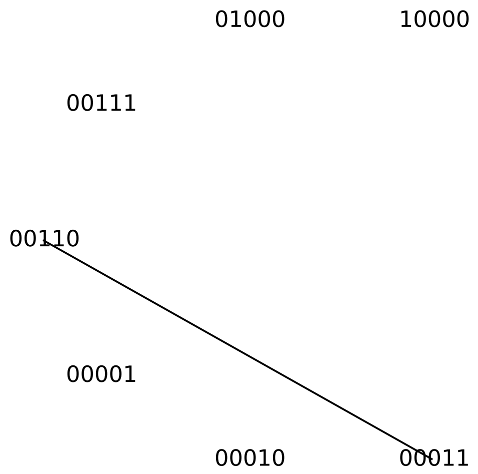
Slika 3.2: Primera 2 in 3 prikazujeta kompatibilni množici ϕ, v primeru 1 pa sta množici nekompatibilni.

velja pogoj 3.5, potem sta binarna niza kompatibilna. Vennov diagram pripadajočih množic $\phi(j)$ in $\phi(k)$ prikazuje slika 3.2. Če velja $|\phi(j)| = 1$ ali $|\phi(k)| = 1$, potem sta binarna niza j in k vedno kompatibilna.

Če kompatibilnost kladov interpretiramo v smislu filogenetskega drevesa, potem posamezni klad predstavlja poddrevo s korenom, ki je notranje vozlišče drevesa T_i. Notranje vozlišče filogenetskega drevesa predstavlja skupnega prednika vsem vozliščem v poddrevesu pod njim. Nekompatibilni sta tisti poddrevesi, ki trdita, da se je iz obeh skupnih prednikov razvilo nekaj skupnih taksonomskih enot, a je hkrati v vsaj enem poddrevesu prisotna taksonomska enota, ki je v drugem poddrevesu ni. V tem primeru poddrevesi torej ponujata nasprotujoči si informaciji o evolucijski zgodovini.

3.3 Graf nekompatibilnosti

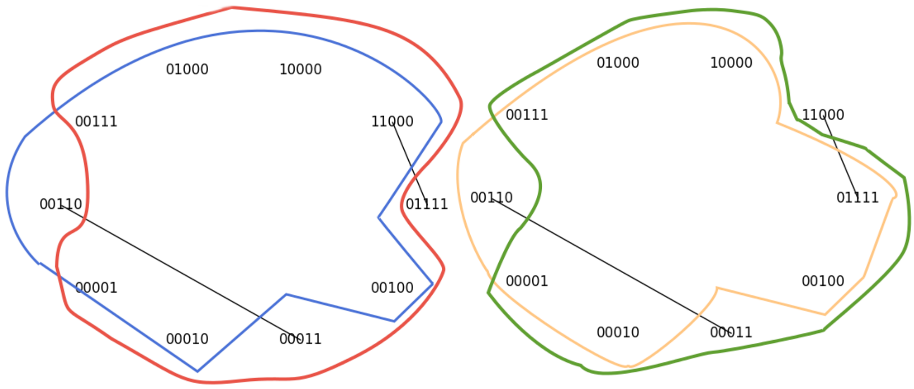
Z uporabo kriterija nekompatibilnosti binarnih nizov lahko zgradimo graf nekompatibilnosti $I(V_1, V_2, ..., V_k, E) = I(V, E)$. Vsaka množica vozlišč V_i predstavlja binarne nize $C(T_i)$, povezave v grafu pa vzpostavimo med tistimi vozlišči, katerih pripadajoči binarni nizi so med seboj nekompatibilni [2]. S povezovanjem vozlišč označimo binarne nize, ki v končnem drevesu skupaj ne morejo biti prisotni, sicer bi filogenetsko drevo podajalo dvoumne informacije



Slika 3.3: Graf nekompatibilnosti bitnih nizov za vhodna drevesa iz slike 2.1. Vozlišča predstavljajo unikatne binarne nize iz vseh dreves vhodne množice T, povezave pa so vzpostavljene med pari vozlišč, katerih binarni nizi medsebojno niso kompatibilni.

o evolucijski zgodovini.

Graf nekompatibilnosti vseh treh vhodnih dreves s slike 2.1 je prikazan na sliki 3.3. V prvem in drugem drevesu sta medsebojno nekompatibilna klada $(Gamma, Alpha)$ in $(Gamma, Beta)$. Drevesi ponujata nasprotujoči si informaciji - prvo za najbolj sorodni taksonomski enoti *Gamma* ponuja taksonomsko enoto *Alpha*, drugo pa taksonomsko enoto *Beta*. Prvo in drugo drevo imata medsebojno kompatibila klada $(Epsilon, Delta)$, ki pa sta nekompatibilna s kladom $(Delta, Gamma, Beta, Alpha)$ iz tretjega vhodnega drevesa. Medtem ko prvo in drugo drevo trdita, da imata taksonomski enoti *Epsilon* in *Delta* neposrednega skupnega prednika, tretje drevo temu nasprotuje.



Slika 3.4: Vse štiri različne največje neodvisne množice za graf nekompatibilnosti s slike 3.3.

3.4 Največja neodvisna množica

Graf nekompatibilnosti vsebuje vse informacije, ki jih potrebujemo za izbiro binarnih nizov, ki bodo prisotni v končnem filogenetskem drevesu. Vanj želimo vključiti kar se da veliko informacij, prisotnih v drevesih vhodne množice T. Najprej je treba zagotoviti, da vključimo binarne nize, ki so skupni vsem vhodnim drevesom T_i (drevo, ki bi vsebovalo le te binarne nize, bi bilo striktno konsenzno drevo). Obenem želimo vključiti čim več preostalih binarnih nizov, vendar ne takih, da bi bil katerikoli par nekompatibilen.

Matematično orodje, ki nam iz grafa $I(V, E)$ omogoča izbiro vozlišč, ki paroma ne bodo kršile pogoja nekompatibilnosti, se imenuje neodvisna množica (angl., independent set). Neodvisna množica grafa je katerakoli množica vozlišč $V_{Indep} \subseteq V$, med katerimi ni medsebojnih povezav. Največja neodvisna množica grafa (angl., maximum independent set - MIS) je tista neodvisna množica $V_{MIS} \subseteq V$, ki vsebuje največ vozlišč. Treba je poudariti, da za neki graf $I(V, E)$ lahko obstaja več različnih V_{MIS}. Ker vozlišča V_{MIS} predstavljajo binarne nize oz. klade končnega drevesa, to pomeni, da za en graf $I(V, E)$ lahko obstaja več asimetričnih srednjih dreves. Primeri vseh največjih neodvisnih množic za graf nekompatibilnosti iz slike 3.3 so narisani na sliki 3.4.

Vozlišča grafa $I(V, E)$, katerega sestavimo iz k dreves vhodne množice T, sestavljajo neodvisne množice $V_1, V_2, ..., V_k$. Vozlišča znotraj ene množice nikoli niso medsebojno povezana, saj njihovi pripadajoči binarni nizi pripadajo istemu drevesu. Vozlišči iz različnih množic pa lahko tvorita povezavo, če sta nekompatibilni. Tak graf imenujemo tudi k-partitni graf. Za bipartitne grafe obstaja trivialen polinomski algoritem s časovno zahtevnostjo $O(n^2)$, v splošnem pa problem največje neodvisne množice za k-partitne grafe sodi v razred NP-težkih algoritmov [2].

3.5 Rekonstrukcija drevesa

S tem, ko smo določili največjo neodvisno množico V_{MIS} grafa $I(V, E)$, smo določili klade končnega drevesa, katerega strukturo pa je treba še rekonstruirati iz izbranih binarnih nizov. Problem rekonstrukcije evolucijske zgodovine n taksonomskih enot iz njihovih binarnih nizov je dobro znan in se imenuje problem filogenije. Zanj obstaja algoritem s časovno kompleksnostjo $O(nm)$, pri čemer je n število taksonomskih enot, m pa število binarnih nizov v največji neodvisni množici [4].

Najprej kreiramo matriko M dimenzij $n \times m$. Binarne nize, ki pripadajo vozliščem množice V_{MIS}, kot stolpce zložimo v matriko M. Nato stolpce interpretiramo kot števila v binarnem zapisu, jih pretvorimo v desetiško obliko in sortiramo. Odstranimo stolpce s podvojeno vrednostjo, tako da ostanejo le stolpci z unikatnimi števili. S tem pridobimo matriko M'.

Naj bo I_i množica indeksov vrstic, I_j pa množica indeksov stolpcev matrike M'. Množica E (3.6) predstavlja vse pare indeksov $(i, j) \in I_i \times I_j$ matrike M', katerih vrednost je enaka ena.

$$E = \{(i, j) : i \in I_i, j \in I_j, M'(i, j) = 1\} \tag{3.6}$$

$$P(i, j) = \begin{cases} \max(\{k : (i, k) \in E, k < j\}) \\ 0 \text{ če tak } k \text{ ne obstaja} \end{cases} \tag{3.7}$$

$$
\begin{array}{l}
\textit{Epsilon} \\ \textit{Delta} \\ \textit{Gamma} \\ \textit{Beta} \\ \textit{Alpha}
\end{array}
\begin{pmatrix}
0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix}
\rightarrow
\begin{array}{c}
i/j \\ 1 \\ 2 \\ 3 \\ 4 \\ 5
\end{array}
\begin{array}{c}
\begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} \\
\begin{pmatrix}
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 1
\end{pmatrix}
\end{array}
= M'
$$

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

$$
P[i,j] =
\begin{array}{c}
i/j \\ 1 \\ 2 \\ 3 \\ 4 \\ 5
\end{array}
\begin{array}{c}
\begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} \\
\begin{pmatrix}
0 & 1 & / & / & / & / & / & / \\
0 & / & 1 & / & / & / & / & / \\
/ & / & / & 0 & 4 & 5 & / & / \\
/ & / & / & 0 & 4 & / & 5 & / \\
/ & / & / & 0 & / & / & / & 4
\end{pmatrix}
\end{array}
\quad
P[j] =
\begin{array}{c}
\begin{array}{ccccccccc} j & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} \\
\begin{pmatrix} 0 & 1 & 1 & 0 & 4 & 5 & 5 & 4 \end{pmatrix}
\end{array}
$$

Slika 3.5: Primer izračuna matrike M', množice parov indeksov E in vseh vrednosti $P(i,j)$ ter $P(j)$ za prvo največjo neodvisno množico s slike 3.4. Ker enačba 3.9 v tem primeru velja, iz matrike M' lahko zgradimo filogenetsko drevo.

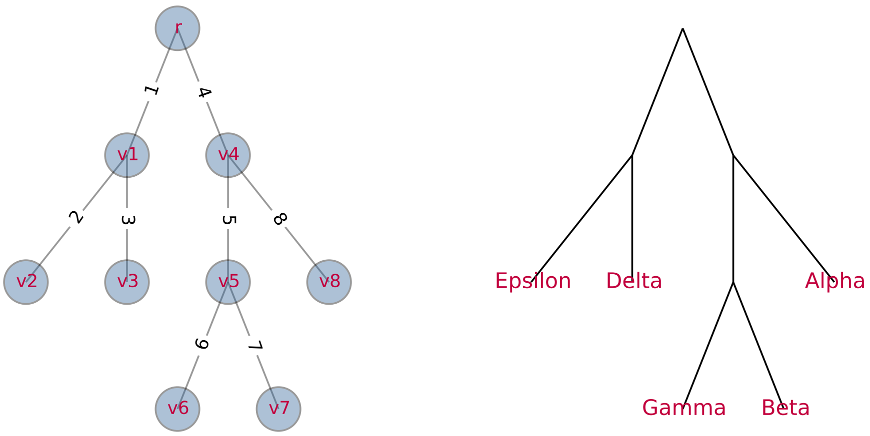
Vsak par $(i,j) \in E$ predstavlja klad j, v katerem je prisotna taksonomska enota i. Za vsak tak par poiščemo vrednost $P(i,j)$, ki ustreza indeksu klada k, v katerem je vsebovan klad j (3.7), pri čemer tudi klad k vsebuje taksonomsko enoto i. Če tak klad ne obstaja, to pomeni, da ima klad j svoj koren pozicioniran najbolj blizu korena celotnega končnega drevesa izmed vseh kladov s prisotno taksonomsko enoto i.

Ko imamo poračunane vse vrednosti $P(i,j)$, lahko za vsak $j \in I_j$ poiščemo vrednost $P(j)$(3.8). Ta predstavlja najvišji indeks klada, v katerem je klad j vsebovan. Ker so kladi sortirane padajoče glede na desetiške vrednosti, najvišji indeks pravzaprav pomeni, da iščemo očeta klada j, ki je na čim nižjem nivoju v končnem drevesu, tako da je klad j še v celoti vsebovan.

$$P(j) = \max(\{P(i,j) : (i,j) \in E\}) \tag{3.8}$$

$$P(i,j) = P(j) \qquad \forall (i,j) \in E \tag{3.9}$$

Če velja enačba 3.9, potem za matriko M' obstaja filogenetsko drevo [4].



Slika 3.6: Na levi strani je prikazan graf, ki je rekonstruiran iz vrednosti $P[j]$ s slike 3.5. Da iz grafa pridobimo končno drevo, prikazano na desni strani, za označevanje listov uporabimo matriko M' s slike 3.5.

V tem primeru se lahko lotimo njegove konstrukcije. Najprej ustvarimo graf, v katerega kot vozlišča dodamo koren bodočega drevesa r in za vsak $j \in I_j$ svoje vozlišče v_j. Nato za vsako vozlišče v_j, za katerega velja $P(j) > 0$, ustvarimo povezavo $(v_{P(j)}, v_j)$ in jo označimo z j. Vsa preostala vozlišča v_j, za katere velja $P(j) = 0$, povežemo s korenom drevesa s pomočjo povezave (r, v_j), in prav tako označimo z j. Primer takega grafa, konstruiranega za matriko M' s slike 3.5 je prikazan na levi strani slike 3.6. Tak graf je že drevo, vendar je potrebno razrešiti oznake na povezavah.

Ker vsaka vrstica matrike M' predstavlja eno taksonomsko enoto, lahko oznake povezav določimo tako, da pregledujemo matriko po vrsticah. Za vsak $i \in I_i$ najdemo največji j, za katerega velja $M'(i, j) = 1$. Dobljeni j nam predstavlja oznako povezave, na koncu katere je prisoten list, ki mu dodelimo oznako taksonomske enote trenutne vrstice i. To storimo za vse vrstice, s čimer označimo vse liste. Kot prikazuje primer dokončanega drevesa na desni strani slike 3.6, za konec notranjim vozliščem in povezavam odstranimo oznake. S tem smo prišli do končnega filogenetskega drevesa, ki ga označimo s τ.

3.6 Vrednost asimetričnega srednjega drevesa

Omenili smo že, da za en graf nekompatibilnosti lahko obstaja več največjih neodvisnih množic in da to pomeni, da lahko sestavimo več različnih asimetričnih srednjih dreves. Prvotni razlog za uporabo konsenznih metod je iz več dreves pridobiti eno, zato potrebujemo metriko, s pomočjo katere se bomo lahko odločili za eno, najbolj podprto drevo.

Najprej uvedemo utež binarnega niza $w(c_e)$ (3.10), ki je preprosto število dreves v vhodni množici T, ki vsebujejo binarni niz c_e. Vrednost asimetričnega srednjega drevesa τ glede na vhodno množico T je določena (3.11) kot vsota uteži binarnih nizov, ki so prisotni v asimetričnem srednjem drevesu, vendar niso prisotni v vseh vhodnih drevesih (oz. v striktnem konsenznem drevesu) [2].

$$w(c_e) = |\{i : c_e \in C(T_i)\}| \tag{3.10}$$

$$value(T, \tau) = \sum_{c \in C(\tau) - \cap C(T_i)} w(c) \tag{3.11}$$

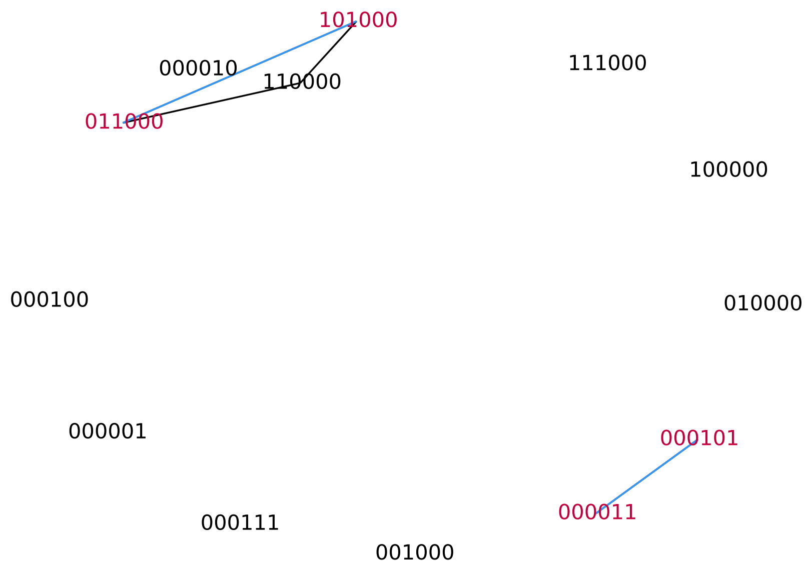
Najboljše najdeno asimetrično srednje drevo je tisto, ki maksimizira dano vsoto uteži 3.11.

3.7 Aproksimacijski algoritmi

Ker je problem konstrukcije asimetričnega srednjega drevesa polinomsko prevedljiv na problem največje neodvisne množice [2] in zato za več kot dve drevesi sodi v razred NP-težkih problemov, je lahko izvajanje algoritma prepočasno, če imamo veliko dreves v vhodni množici T. Zato si bomo ogledali dva aproksimacijska algoritma, ki imata polinomsko časovno zahtevnost.

Prvi, t.j. $\frac{2}{k}$-aproksimacijski algoritem, je enostaven. Za vsak par dreves $T_i, T_j \in T$ izračunamo asimetrično srednje drevo τ_{ij} in izberemo tistega, ki maksimizira vrednostno funkcijo $value(T, \tau_{ij})$. Ker je tako asimetrično srednje drevo sestavljeno iz dveh dreves, posledično lahko vsebuje le klade, ki so prisotni v teh dveh drevesih.

Če imajo drevesa vhodne množice T veliko skupnih kladov, je primernejši drugi algoritem. Naj bo $V^* \subseteq V$ podmnožica vozlišč grafa nekompatibilnosti $I(V, E)$, in sicer tistih, ki so skupna vsem drevesom. Takih vozlišč v naslednjem koraku ne potrebujemo, saj niso povezana z nobenim drugim vozliščem. Nato izračunamo največje ujemanje (angl., maximum matching) v grafu $I(V - V^*, E)$, s čimer pridobimo množico robov $E_M \subseteq E$, ki nimajo skupnih vozlišč [24]. Vozlišča V_M, ki so krajišča robov iz množice E_M, odstranimo iz množice V in ostanejo nam le neujemajoča vozlišča $V_N = V - V_M$. Primer iskanja množice V_N je prikazan na sliki 3.7. Iz binarnih nizov, pripadajočih vozliščem množice V_N, rekonstruiramo asimetrično srednje drevo po običajnem postopku. Tako drevo, za razliko od drevesa izračunanega s pomočjo prvega aproksimacijskega algoritma, lahko vsebuje klade iz več dreves vhodne množice hkrati [2].



Slika 3.7: Primer računanja bitnih nizov asimetričnega srednjega drevesa s pomočjo največjega ujemanja. Robovi, ki spadajo v množico največjega ujemanja E_M, so obarvani modro, njihova krajišča pa rdeče. Vsi bitni nizi, ki so obarvani črno, sestavljajo končno drevo.

Poglavje 4

Implementacija algoritma

4.1 Biopython

Biopython [17] je odprtokodni paket modulov, napisan v programskem jeziku Python, ki ponuja mnoge funkcionalnosti s področja bioinformatike. Olajša nam delo s formati datotek, kot so BLAST, ClustalW, FASTA ter GenBank in ponuja enostaven dostop do spletnih storitev, npr. NCBI. Poleg funkcionalnosti, implementiranih v programskem jeziku Python, ponuja tudi enostaven dostop do zunanjih programov. Nekateri glavni moduli paketa Biopython, ki so relevantni tudi za računanje filogenetskih dreves, so:

- *Bio.SeqIO*: razredi, ki nam omogočajo branje, pisanje in manipuliranje sekvenc v različnih formatih (*FASTA*, *Nexus*, ...),
- *Bio.AlignIO*: razredi, ki nam omogočajo branje, pisanje in manipuliranje poravnanih sekvenc,
- *Bio.Application*: razredi, s pomočjo katerih enostavno dostopamo do zunanje namenske programske opreme, kot sta PhyML in RAxML, in
- *Bio.Phylo*: razredi in funkcije za izračun filogenetskih dreves.

Modul *Bio.Phylo* že omogoča izračun filogenetskih dreves s pomočjo metod UPGMA, Neighbor Joining in Maximum Parsimony. Poleg tega ponuja tudi metode ponovnega vzorčenja bootstrap in jackknife.

Izračun konsenznih filogenetskih dreves je mogoč s pomočjo podmodula *Bio.Phylo.Consensus*, v katerem so bile najprej implementirane metode striktnega, večinskega in adamovega konsenza, zdaj pa izračun lahko opravimo tudi s pomočjo metode asimetričnega srednjega drevesa. Branje in pisanje dreves je že omogočeno za formate Newick, CDAO in PhyloXML. Modul ponuja tudi vmesno reprezentacijo drevesa, ki ga uporabljamo med izračunom in ni vezan na končni format.

4.2 Podrobnosti implementacije

Glavna entiteta, ki je nastopala v obravnavi metode asimetričnega srednjega drevesa, je bil bitni niz. Zato je pomembno, da zanj uporabimo primerno hitro podatkovno strukturo. Razred, ki podatkovno strukturo že implementira, se imenuje *_BitString*, prebiva pa v podmodulu *Bio.Phylo.Consensus*. *_BitString* razširja Pythonovo vgrajeno podatkovno strukturo *String*, zato ga lahko uporabljamo na enak način. Med operatorji, ki jih lahko izvajamo nad podatkovno strukturo, so konjunkcija, disjunkcija in ekskluzivna disjunkcija nad posameznimi mesti, vsebuje pa tudi metodi za preverjanje vsebovanosti enega bitnega niza v drugem ter preverjanje kompatibilnosti dveh binarnih nizov.

4.2.1 Izračun največje neodvisne množice

Modul *Bio.Phylo* za izrisovanje filogenetskih dreves uporablja paket networkx[1]. Ta poleg samega izrisovanja ponuja tudi konstrukcijo grafov in njihovo nadaljnjo manipulacijo. Na grafu zmore izračunati aproksimacijo največje neodvisne množice, a se s tem nismo mogli zadovoljiti, saj je bila aproksimacija že za majhna vhodna drevesa preveč nenatančna, da bi dobili uporaben rezultat. Na srečo je problem največje neodvisne množice polinomsko prevedljiv na problem največje klike v komplementarnem grafu, ki

[1] `https://networkx.github.io/`

jo networkx zna poiskati. Kljub temu se je iskanje vseh največjih klik komplementarnega grafa izkazalo za dokaj počasno.

Ob opazovanju, da naš algoritem izrablja le eno procesorsko jedro, smo se odločili, da lahko iskanje največje klike opravimo na vseh razpoložljivih procesorjih s pomočjo zunanjega programa Parallel Maximum Clique (PMC) [20], če zaznamo prisotnost programa na uporabnikovem sistemu. Čeprav smo s tem vnesli dodatno odvisnost od zunanje programske opreme, lahko odločitev utemeljimo s precej večjo hitrostjo implementacije PMC v primerjavi z lastno implementacijo v programskem jeziku Python. Če program ni najden, se iskanje največjih klik kljub temu opravi z uporabo paketa networkx.

4.2.2 Izračun največjega ujemanja v grafu

Pri uporabi druge aproksimacijske metode za izračun asimetričnega srednjega drevesa ne računamo največje neodvisne množice, temveč največje ujemanje grafa. Paket networkx za reševanje problema vsebuje funkcijo *max_weight_matching()*, pri čemer so vse uteži robov grafa enake 1.

Tako kot natančna metoda, tudi prva aproksimacijska metoda v grafu nekompatibilnosti išče največjo neodvisno množico. V kolikor bi za obe metodi uporabili isti algoritem za iskanje največje neodvisne množice, bi bila prva aproksimacijska metoda mnogo počasnejša kot je sicer lahko. Zato smo izkoristili dejstvo, da graf nekompatibilnosti v tem primeru sestavljata le dve particiji. Za tak graf obstaja polinomski algoritem [22], s pomočjo katerega lahko izračunamo največje ujemanje. Problem največjega ujemanja je v polinomskem času prevedljiv v problem minimalnega pokritja bipartitnega grafa, ta pa v bipartitnih grafih predstavlja komplement maksimalne neodvisne množice. Programsko kodo, s katero smo največje ujemanje bipartitnega grafa prevedli v vozlišča največje neodvisne množice, prikazuje slika 4.1.

```python
verts = graph.nodes()
top, bottom = nx.bipartite.sets(graph)
mates = nx.max_weight_matching(graph, maxcardinality=True)
matched = mates.keys()
top_unmatched = [v for v in top if v not in matched]
cwo = [
    adj for adj in graph[v] if adj in bottom
    for v in top_unmatched
]
cwo += top_unmatched
# minimalno pokritje
mvc = set(
    [v for v in top if v not in cwo]
    + list(set(bottom) & set(cwo))
)
# maksimalna neodvisna mnozica
mis = [v for v in verts if v not in mvc]
```

Slika 4.1: Programska koda za prevedbo največjega ujemanja v bipartitnem grafu v minimalno pokritje grafa in nato v maksimalno neodvisno množico.

4.2.3 Rekonstrukcija drevesa

Za rekonstrukcijo drevesa smo potrebovali orodje, ki je zmožno dela z matrikami. Ker je paket Biopython že odvisen od paketa numpy[2], je ta bil logična izbira. Med kreiranjem matrike je bilo treba binarne nize pretvoriti v stolpce matrike. Za to potrebo smo razredu _BitString dodali metodo *to_numpy*(), ki niz ničel in enic pretvori v binarni vektor tipa *numpy.ndarray* z eno dimenzijo. Za potrebe sortiranja smo razredu *_BitString* dodali še metodo *__int__*(), ki izračuna desetiško vrednost binarnega niza.

[2] http://www.numpy.org/

4.2.4 Programski vmesnik

Slika 4.2 prikazje glavo implementirane funkcije, ki je v modulu *Bio.Phylo.Consensus*. Celotna koda je navedena v dodatku B. Metoda privzeto izračuna točno asimetrično srednje drevo, vendar je računska kompleksnost za več kot dve drevesi eksponentna, zato lahko uporabnik, če to želi, z drugim parametrom izbere eno izmed aproksimacijskih metod.

```
def amt_consensus(trees, method='no_approx')
```

Slika 4.2: Glava funkcije za izračun asimetričnega srednjega drevesa iz modula *Bio.Phylo.Consensus*.

Metoda *Bio.Phylo.Consensus.amt_consensus* sprejema dva parametra:

- **trees** je seznam objektov tipa *BaseTree.Tree*. Drevo je lahko v kateremkoli formatu, ki ga podpira Biopython.
- **method** je tipa *niz*, s katerim izbiramo metodo izračuna. Veljavne so tri vrednosti:
  - **no_approx** izračun točnega drevesa,
  - **bi_approx** izračun s kombiniranjem največ dveh dreves,
  - **maxmatch_approx** izračun drevesa s pomočjo največjega ujemanja.

4.3 Enostaven primer uporabe

Primer uporabe programske kode je najbolj enostavno prikazati najprej na enostavnejšem primeru. Recimo, da imamo pripravljeno datoteko z imenom *primer_a.tre* in vsebino na sliki 4.3. Vsebuje pet dreves v formatu Newick, iz katerih želimo izračunati asimetrično srednje drevo in ga izrisati.

Uporaba konsenzne metode asimetričnega srednjega drevesa v paketu Biopython je enostavna. Slika 4.4 predstavlja osnovni primer uporabe. V prvi

```
((A, (B, C)), (D, (E, F)));
((B, (A, C)), (D, (E, F)));
((A, (B, C)), (E, (D, F)));
((B, (A, C)), (E, (D, F)));
((C, (B, A)), (E, (D, F)));
```

Slika 4.3: Primer petih polno razrešenih filogenetskih dreves s šestimi taksonomskimi enotami, ki so kodirana v formatu Newick.

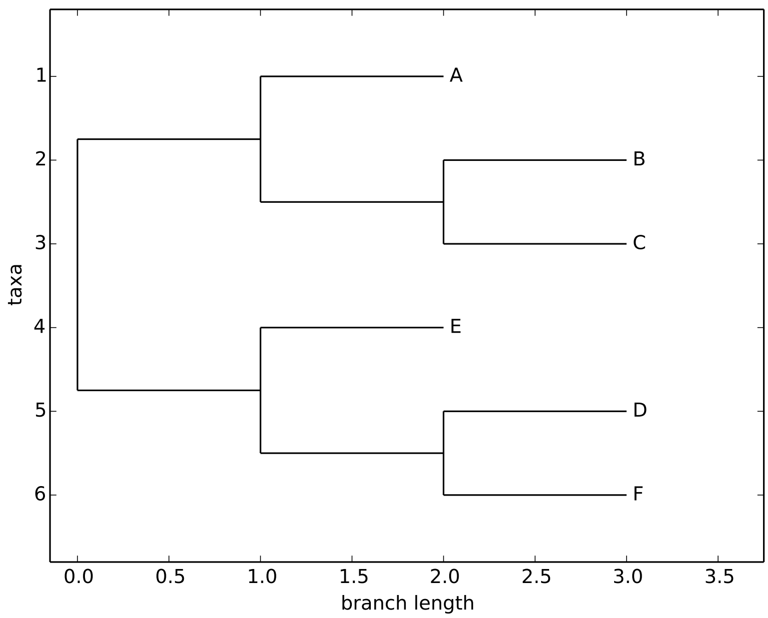
vrstici uvozimo modul *Bio.Phylo*, v drugi iz podmodula *Bio.Phylo.Consensus* uvozimo funkcijo *amt_consensus*, v tretji pa uvozimo paket za risanje *pyplot*.

```
1 from Bio import Phylo
2 from Bio.Phylo.Consensus import amt_consensus
3 from matplotlib import pyplot
4
5 trees = list(Phylo.parse('primer_a.tre', 'newick'))
6 amt = amt_consensus(trees)
7 amt.ladderize()
8 Phylo.draw(amt)
9 pyplot.show()
```

Slika 4.4: Primer programske kode za branje dreves iz datoteke in izračun asimetričnega srednjega drevesa.

V peti vrstici uvozimo filogenetska drevesa iz datoteke *primer_a.tre* in jih razčlenimo, s čimer dobimo seznam objektov tipa *Bio.Phylo.BaseTree.Tree*. V šesti vrstici s preprostim klicem funkcije iz obstoječega seznama dreves izračunamo asimetrično srednje drevo. V sedmi vrstici drevo preoblikujemo tako, da bodo globlja poddrevesa prikazana na vrhu, in v zadnjih dveh vrsticah drevo izrišemo na ekran.

Rezultat programa iz figure 4.4 je prikazan na sliki 4.5. Poleg drevesa so prikazane še osi, ki v našem primeru sicer niso pomembne, v splošnem pa iz osi x lahko razberemo divergentne čase taksonomskih enot.



Slika 4.5: Rezultat programa iz slike 4.4.

4.4 Primer uporabe z grajenjem dreves iz sekvenc DNA

V prejšnjem primeru uporabe smo filogenetska drevesa prebrali iz vnaprej pripravljene datoteke. Običajno ni tako in je drevesa pred uporabo konsenzne metode treba najprej izračunati. Primer programske kode, s katero v paketu Biopython izračunamo filogenetska drevesa s pomočjo metod UPGMA, metode združevanja sosedov in metode največje varčnosti, iz njih pa tri konsenzna drevesa, prikazuje slika 4.7.

Najprej v šesti vrstici s pomočjo modula *Bio.AlignIO* iz vhodne datoteke uvozimo vnaprej poravnane sekvence DNA, ki pripadajo taksonomskim enotam. Nato v deseti vrstici kreiramo objekt tipa *DistanceCalculator*, ki služi za

```
        5  13
Alpha      AACGTGGCCACAT
Beta       AAGGTCGCCACAC
Gamma      CAGTTCGCCACAA
Delta      GAGATTTCCGCCT
Epsilon    GAGATCTCCGCCC
```

Slika 4.6: Vhodne sekvence DNA v datoteki *msa.phy*, ki je ena izmed testnih datotek programskega paketa Biopython [18], v formatu, ki ga uporablja programski paket PHYLIP.

izračun distančne matrike. Tega potrebujemo v naslednjih dveh vrsticah, kjer kreiramo konstruktor dreves tipa *DistanceTreeConstructor* za metodo UPGMA in metodo združevanja najbližjih sosedov. Metoda največje varčnosti za razliko od distančnih metod potrebuje objekt tipa *ParsimonyScorer*, ki ga kreiramo v petnajsti vrstici, in preiskovalnik prostora dreves tipa *NNITreeSearcher*, ki ga kreiramo vrstico nižje. Oba objekta podamo konstruktorju metode največje varčnosti v sedemnajsti vrstici.

Zdaj imamo pripravljeno vse, da lahko v vrsticah 20-24 s pomočjo prej ustvarjenih konstruktorjev dreves iz vhodnih sekvenc DNA zgradimo tri filogenetska drevesa. Nato v vrsticah 27-29 iz zgrajenih dreves izračunamo še konsenzna drevesa s pomočjo metod asimetričnega srednjega drevesa, večinskega konsenza in adamovega konsenza. Tako kot v prvem primeru uporabe za konec izračunana konsenzna drevesa še izrišemo, kar naredimo s pomočjo paketa *pyplot*.

```python
from Bio import AlignIO
from Bio import Phylo
from Bio.Phylo.Consensus import *
from Bio.Phylo.TreeConstruction import *
from matplotlib import pyplot

input_file = './biopython/Tests/TreeConstruction/msa.phy'
alignments = AlignIO.read(open(input_file), 'phylip')
# Objekti za izracun distancne matrike,
# UPGMA in NeighborJoining konstruktor
calculator = DistanceCalculator('identity')
nj_constr = DistanceTreeConstructor(calculator, 'nj')
upgma_constr = DistanceTreeConstructor(calculator, 'upgma')
# Objekti za izracun najbolj varcnega drevesa
scorer = ParsimonyScorer()
searcher = NNITreeSearcher(scorer)
pars_constr = ParsimonyTreeConstructor(searcher)
# Izracunamo filogenetska drevesa
trees = [
    nj_constr.build_tree(alignments),
    upgma_constr.build_tree(alignments),
    pars_constr.build_tree(alignments)
]
# Izracunamo konsenzna filogenetska drevesa
amt = amt_consensus(trees, method='no_approx')
majority = majority_consensus(trees)
adam = adam_consensus(trees)
for i, t in enumerate([amt, majority, adam]):
    f = pyplot.figure(i)
    f.clf()
    Phylo.draw(t)
    pyplot.show()
```

Slika 4.7: Primer programske kode za konstrukcijo konsenznih filogenetskih dreves v programskem paketu Biopython. Konsenzna drevesa izračunamo iz treh filogenetskih dreves, ki jih izračunamo s pomočjo metode UPGMA, metode združevanja najbližjih sosedov in metode največje varčnosti iz vhodnih sekvenc DNA.

Poglavje 5

Eksperimentalna primerjava

Vse tri implementirane metode računanja asimetričnega srednjega drevesa želimo primerjati med seboj in glede na metode striktnega, večinskega ter adamovega konsenza. Za ta namen smo pripravili tri vhodne množice in en dejanski primer, na katerih smo ovrednotili algoritme glede na razrešenost končnega drevesa ter glede na Robinson-Fouldsovo metriko. Programska koda, s katero smo pridobili rezultate, je v dodatku A. Poleg lastnosti izhodnih dreves nas je zanimal tudi čas izvajanja vseh treh implementiranih metod, s čimer smo ocenili velikost vhodne množice in število taksonomskih enot v drevesih, za katere zaradi predolgega časa izvajanja ni več smiselno uporabiti natančne metode. Tudi koda za merjenje časa je v dodatku A.

5.1 Robinson-Fouldsova mera razrešenosti drevesa

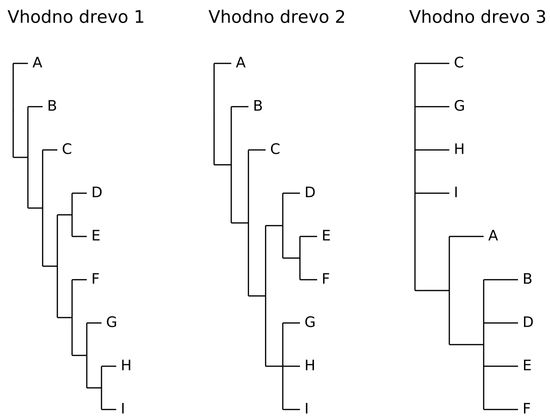
Razrešenost drevesa (5.1) je definirana kot število povezav, ki nastopajo v drevesu. Ker imajo vsa drevesa enako število listov, se je drevo z največjim številom povezav največkrat razvejalo. Najbolj razrešeno drevo je binarno drevo (takrat rečemo, da je *polno razrešeno*). To ponuja največ informacij o evolucijski zgodovini, saj za vsak par taksonomskih enot vemo, iz katerega skupnega prednika sta se enoti razvili. Zaželeno je torej, da je število povezav v drevesu čim večje.

$$Res(T) = |E(T)| \tag{5.1}$$

$$RF(T_1, T_2) = |C(T_1) - C(T_2)| + |C(T_2) - C(T_1)| \tag{5.2}$$

Robinson-Fouldsova (RF) metrika je najbolj razširjena metrika za primerjanje filogenetskih dreves. Šteje število kladov (5.2), ki si jih drevesi ne delita [8]. Za izračun vrednosti RF smo uporabili funkcijo *symmetric_difference()* iz paketa DendroPy[1], saj v paketu Biopython metrika RF še ni implementirana. Vrednost RF smo izračunali za vsak par konsenznega in vhodnega drevesa, ter za vsako konsenzno drevo sešteli vrednosti RF.

5.2 Vhodna množica 1



Slika 5.1: Prva vhodna množica dreves.

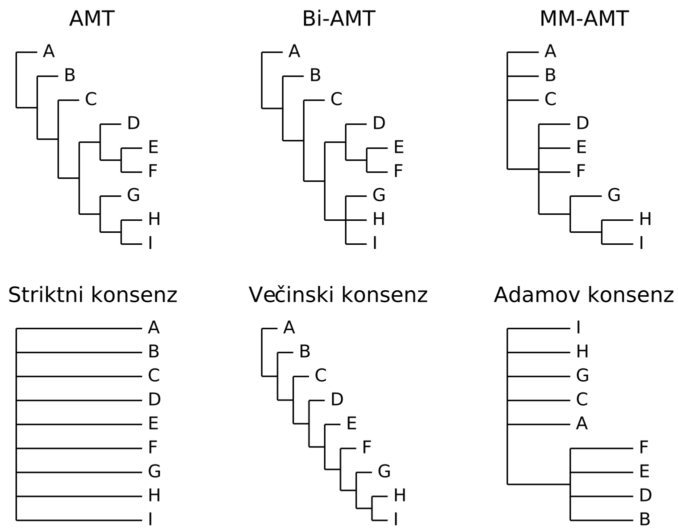
Tabela 5.2 prikazuje razrešenost in vrednosti RF asimetričnega srednjega drevesa, dveh aproksimiranih asimetričnih srednjih dreves (Bi-AMT in MM-AMT), striktnega, večinskega in adamovega konsenznega drevesa s slike 5.2

[1] `https://pythonhosted.org/DendroPy/`

	Res (T)	RF (T_1)	RF (T_2)	RF (T_3)	RF (vsota)
AMT	17	4	1	8	13
Bi-AMT	16	5	0	7	12
MM-AMT	13	3	4	5	12
Striktni konsenz	10	6	5	2	13
Večinski konsenz	17	2	5	4	11
Adamov konsenz	11	5	6	1	12

Tabela 5.1: Razrešenost konsenznih dreves in njihove vrednosti RF za vhodna drevesa na sliki 5.1.

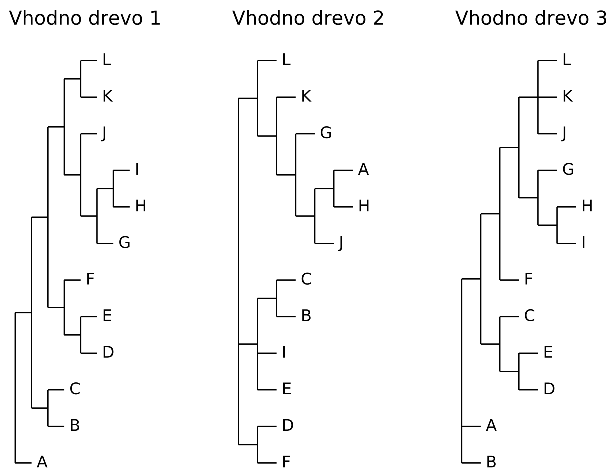
glede na vhodna drevesa s slike 5.1. Opazimo, da je asimetrično srednje drevo glede na razrešenost prav tako dobro kot večinsko konsenzno drevo. Obe drevesi sta polno razrešeni. Glede na vrednosti RF je vhodnim drevesom najbolj podobno večinsko konsenzno drevo. Tako lahko brez dvoma večinsko konsenzno drevo v tem primeru razglasimo za najboljše.



Slika 5.2: Konsenzna drevesa, zgrajena iz množice dreves na sliki 5.1. Oznaka Bi-AMT označuje približek asimetričnega srednjega drevesa, zgrajenega iz dveh vhodnih dreves, MM-AMT pa aproksimirano drevo, zgrajeno s pomočjo največjega ujemanja.

Rezultat je pričakovan, saj je večina kladov prisotnih v več kot polovici vhodnih dreves, preostali pa so povečini medsebojno kompatibilni, kar je za večinsko drevo ugodno. Najbolj podobno asimetrično srednje drevo je MM-AMT, vendar je njegova razrešenost med najslabšimi. Drevo Bi-AMT ne podaja informacije o skupnem predniku taksonomskih enot H in I, zaradi česar je bolj podobno drugemu in tretjemu vhodnemu drevesu kot drevo AMT, posledično pa je njegova razrešenost zaradi tega slabša.

5.3 Vhodna množica 2



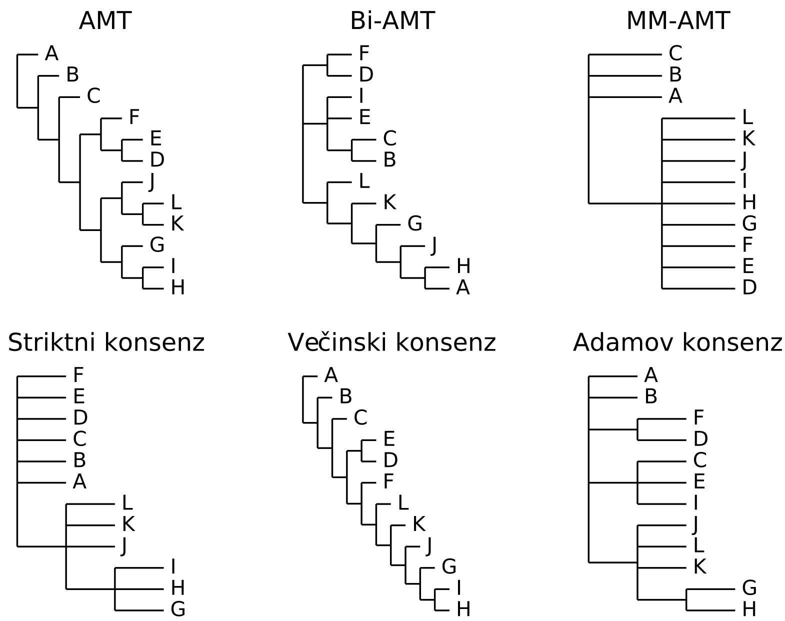
Slika 5.3: Druga vhodna množica dreves.

Rezultati konsenznih dreves s slike 5.4 glede na vhodna drevesa s slike 5.3 so prikazani v tabeli 5.3. Ponovno sta asimetrično srednje drevo ter večinsko konsenzno drevo polno razrešeni. Če ti dve drevesi medsebojno primerjamo še po podobnosti vhodnim drevesom, se je najbolje odrezalo asimetrično srednje drevo. Nekoliko slabše razrešeno je drevo Bi-AMT, ki je sicer glede na metriko RF nekoliko bolj podobno vhodnim drevesom kot asimetrično srednje drevo. Drevo MM-AMT je sicer po podobnosti še boljše, vendar je manj podobno in za povrh še manj razrešeno kot striktno konsen-

	Res (T)	RF (T_1)	RF (T_2)	RF (T_3)	RF (vsota)
AMT	23	12	9	7	28
Bi-AMT	21	9	10	8	27
MM-AMT	14	10	9	7	26
Striktni konsenz	15	7	8	8	23
Večinski konsenz	23	10	11	9	30
Adamov konsenz	17	13	10	8	31

Tabela 5.2: Razrešenost konsenznih dreves in njihove vrednosti RF za vhodna drevesa na sliki 5.3.

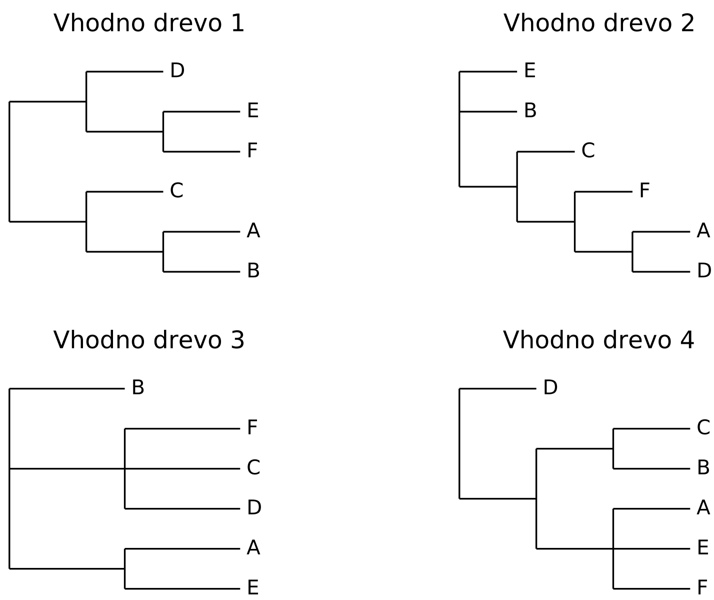
zno drevo. Adamovo konsenzno drevo je kljub slabši razrešenosti najmanj podobno vhodnim drevesom.



Slika 5.4: Konsenzna drevesa, zgrajena iz množice dreves na sliki 5.3.

Ker večina kladov ni prisotnih v več kot polovici vhodnih dreves, povečini pa so medsebojno kompatibilni, je rezultat pričakovan. Slaba razrešenost drevesa MM-AMT je najverjetneje posledica majhnega števila skupnih kladov in majhnega števila vhodnih dreves.

5.4 Vhodna množica 3

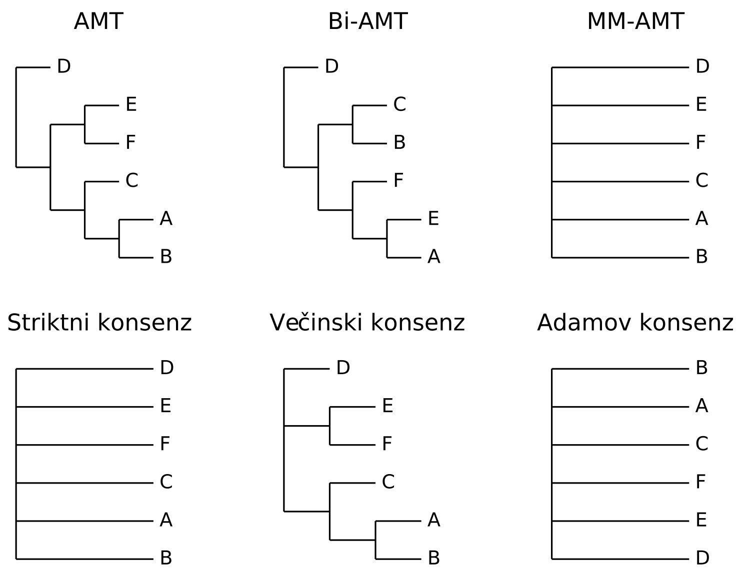


Slika 5.5: Tretja vhodna množica dreves.

Tretja vhodna množica dreves s slike 5.4 je sestavljena tako, da je število skupnih kladov majhno, posamezni kladi ne nastopajo v več kot polovici vhodnih dreves in so večinoma medsebojno nekompatibilni. Rezultate dobljenih konsenznih dreves na sliki 5.5 prikazuje tabela 5.4.

	Res (T)	RF (T_1)	RF (T_2)	RF (T_3)	RF(T_4)	RF (vsota)
AMT	11	0	2	3	1	6
Bi-AMT	11	0	2	3	1	6
MM-AMT	7	3	3	2	2	10
Striktni konsenz	7	3	3	2	2	10
Večinski konsenz	10	0	2	3	1	6
Adamov konsenz	7	3	3	2	2	10

Tabela 5.3: Razrešenost konsenznih dreves in njihove vrednosti RF za vhodna drevesa na sliki 5.5.



Slika 5.6: Konsenzna drevesa, zgrajena iz množice dreves na sliki 5.5.

Opazimo, da sta polno razrešeni le asimetrično srednje drevo in aproksimirano drevo Bi-AMT. Vsa ostala drevesa vsebujejo vsaj eno nerazrešeno vozlišče. Drevo MM-AMT je v tem primeru enako striktnemu konsenznemu drevesu in adamovem konsenznemu drevesu, tako po topologiji kot glede na razrešenost in metriko RF. Preostala drevesa so glede na metriko RF sicer enakovredna, vendar ker drevo večinskega konsenza ni polno razrešeno, lahko za najboljši drevesi razglasimo asimetrično srednje drevo in drevo Bi-AMT, ki sicer nimata popolnoma enakih topologij.

Slaba razrešenost drevesa MM-AMT je ponovno posledica majhnega števila skupnih kladov in v takem primeru je aproksimacija Bi-AMT boljša. Ker posamezni kladi ne nastopajo v več kot polovici vhodnih dreves, metodama striktnega in adamovega konsenza ni uspelo popolnoma razrešiti vseh vozlišč vhodnih dreves, saj so kladi s premalo pojavitvami kaznovani. Asimetrično srednje drevo za razliko take klade vključi, če to prispeva k informiranosti drevesa.

Razlika med asimetričnim srednjim drevesom in aproksimiranim drevesom Bi-AMT ni tako očitna, ker je vhodna množica majhna. Drevo Bi-AMT je namreč zgrajeno le iz dveh vhodnih dreves in če bi vhodno množico povečali, bi postale razlike bolj očitne, saj bi bilo asimetrično srednje drevo za razliko od približka sposobno vključiti informacije iz več dreves hkrati.

5.5 Primer puščavskih zelenih alg

Zadnjo vhodno množico predstavlja deset dreves s 150 taksonomskimi enotami iz zbirke primerov programa HashCS [19]. Zbirka je sicer bila zgrajena za potrebe uvrstitve devetih prej še neobjavljenih sekvenc puščavskih zelenih alg. Poleg njih vsebuje še štirinajst znanih puščavskih zelenih alg in 127 taksonomskih enot prisotnih v tekoči vodi, morski vodi ali tleh. Drevesa so bila zgrajena s pomočjo programa MrBayes z uporabo evolucijskega modela GTR. Poravnane sekvence, ki so bile vhod v metodo bayesove inference, so bile dolge 1651 baznih parov [21].

	AMT	Bi-AMT	MM-AMT	Striktni k.	Večinski k.	Adamov k.
Res(T)	298	279	242	151	284	259
RF(T_1)	238	233	206	147	220	225
RF(T_2)	260	255	204	147	232	215
RF(T_3)	252	251	218	147	244	231
RF(T_4)	258	243	190	147	248	233
RF(T_5)	248	245	208	147	234	219
RF(T_6)	258	229	206	147	232	221
RF(T_7)	266	216	216	147	242	233
RF(T_8)	256	241	212	147	242	221
RF(T_9)	250	243	196	147	232	231
RF(T_{10})	260	233	212	147	232	213
RF (vsota)	2546	2389	2068	1470	2358	2242
Čas [s]	293	164	11.5	0.3	3.5	15

Tabela 5.4: Razrešenost konsenznih dreves in njihove vrednosti RF za deset filogenetskih dreves s 150 taksonomskimi enotami [21].

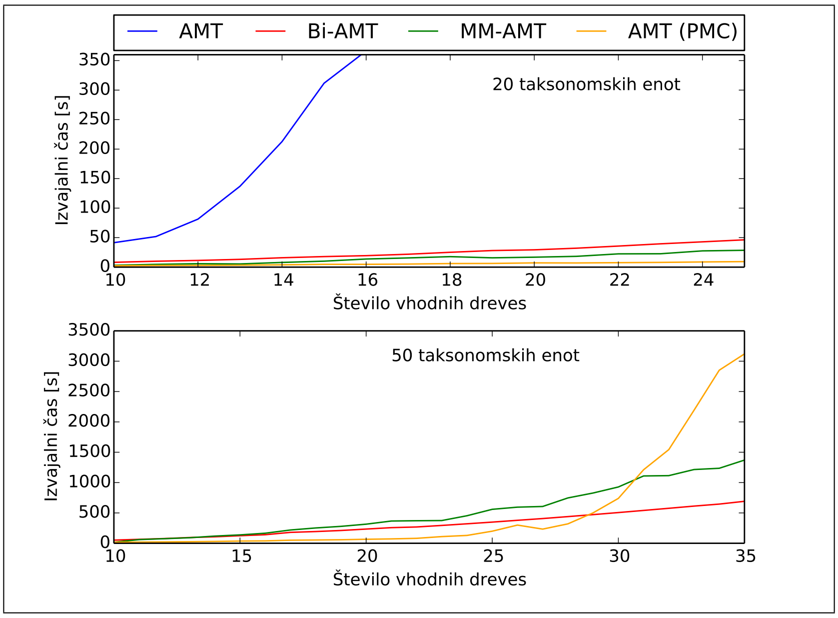
Rezultati v tabeli 5.5 kažejo, da je najbolje razrešeno asimetrično srednje drevo. Sledita mu večinsko konsenzno drevo in aproksimirano drevo Bi-AMT. Striktno konsenzno drevo je sicer najbolj podobno vhodnim drevesom, vendar najslabše razrešeno. Opazimo, da so vsa drevesa do neke mere žrtvovala podobnost za razrešenost (z izjemo drevesa Bi-AMT, ki je slabše razrešeno in manj podobno vhodnim drevesom kot večinsko). V tem smislu je metoda striktnega konsenza najbolj konservativna, metoda asimetričnega srednjega drevesa pa najbolj drzna, s čimer podaja največ informacij o evolucijskih dogodkih.

Časovno se je najbolje odrezala metoda striktnega konsenza, sledi pa ji metoda večinskega konsenza. Aproksimacijska metoda MM-AMT je bila hitrejša od Adamovega konsenza, najpočasnejša pa je bila natančna metoda.

5.6 Primerjava izvajalnih časov

Zaradi eksponentne časovne kompleksnosti metode asimetričnega srednjega drevesa nas zanima, kako velika drevesa in vhodne množice lahko še obdelamo v doglednem času. Merjenje časa izvajanja smo opravili za natančno metodo, obe aproksimacijski metodi in tudi natančno metodo z nameščenim programom PMC. Numerične vrednosti meritev so prikazane v tabeli 5.6.

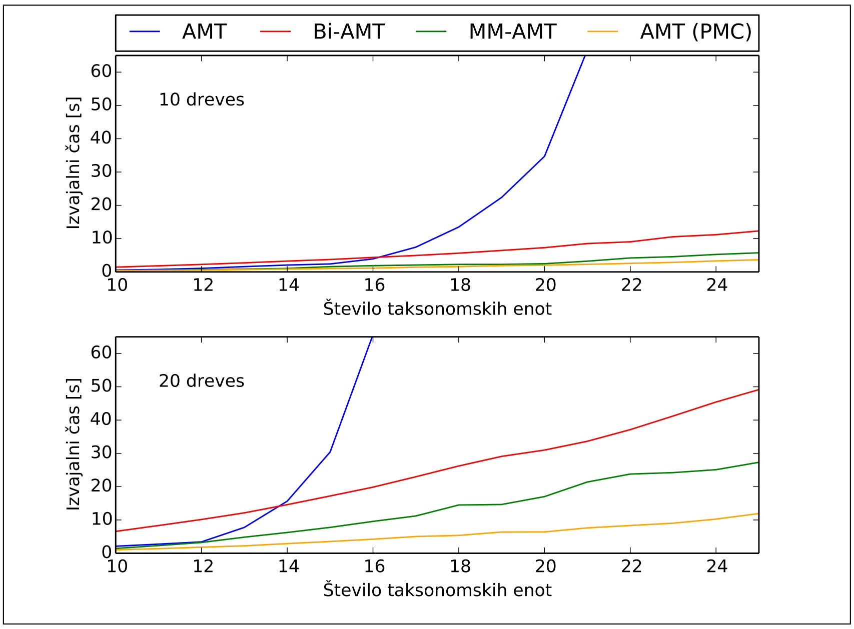
Zgornji graf na sliki 5.7 prikazuje čase izvajanja za drevesa, ki imajo dvajset taksonomskih enot. Opazimo, da se čas izvajanja z večanjem števila vhodnih dreves za natančno metodo AMT povečuje zelo hitro. Preostale metode s takimi drevesi nimajo večjih težav. Spodnji del slike prikazuje graf za drevesa s 50 taksonomskimi enotami. Opazimo, da natančna metoda s pomočjo programa PMC brez težav opravi z manjšimi vhodnimi množicami dreves, nato pa prične čas naraščati precej hitreje. Čas, potreben za izvedbo aproksimacijskih metod tudi z večanjem števila vhodnih dreves pri večjem številu taksonomskih enot narašča polinomsko. Metoda Bi-AMT je kljub hitrejšemu naraščanju izvajalnega časa za manjše število taksonomskih enot pri večjem številu taksonomskih enot hitrejša od metode MM-AMT.



Slika 5.7: Izvajalni časi vseh treh metod za drevesa z 20 in 50 taksonomskimi enotami za vse tri implementirane metode izračuna asimetričnega srednjega drevesa.

Slika 5.8 prikazuje grafa časov izvajanja glede na število taksonomskih enot pri fiksnem številu dreves. Zgornji graf prikazuje izvajalne čase za deset vhodnih dreves, drugi pa za dvajset. Pri obeh so razmerja enaka. Potreben čas za izvedbo natančne metode skokovito poraste, medtem ko čas izvajanja preostalih metod narašča precej počasneje. Najhitrejši je izračun točnega asimetričnega srednjega drevesa s pomočjo programa PMC.

Iz zgornjih grafov je razvidno, da na čas izvajanja bistveno bolj kot število taksonomskih enot vpliva število vhodnih dreves. Vendar ni tako enostavno. Zaradi naključnega generiranja dreves vhodne množice težko trdimo, kakšne so bile njihove lastnosti. Število taksonomskih enot v drevesu je neposredno povezano z globino drevesa. V kolikor je taksonomskih enot več, potem je število možnih poddreves večje. Z večjim številom možnih poddreves se poveča tudi število potencialnih vozlišč v grafu nekompatibilnosti. Slednje je sicer odvisno tako od števila dreves v vhodni množici kot njihovih lastnosti.



Slika 5.8: Izvajalni časi vseh treh metod za vhodne množice 10 ali 20 dreves.

Drevesa lahko vsebujejo veliko število taksonomskih enot, a so si medsebojno precej podobna, pri čemer bo graf nekompatibilnosti majhen, četudi je število vhodnih dreves zelo veliko, in posledično bo čas izvajanja krajši. Na drugi strani število taksonomskih enot ne rabi biti zelo veliko, da naletimo na težave s časom izvajanja, če so drevesa medsebojno zelo različna.

Zaključimo torej lahko, da je izračun asimetričnega srednjega drevesa s pomočjo natančne metode za večje vhodne množice smotrn le z uporabo zunanjega programa PMC. Tudi tu sicer naletimo na težave, vendar precej pozneje. Primer puščavskih zelenih alg, ki vsebuje 150 taksonomskih enot in deset dreves, smo naprimer uspeli izračunati v petih minutah, medtem ko so naključno generirani primeri pokazali, da v splošnem čas, potreben za izračun, lahko skokovito naraste. Ocenimo lahko, da je uporaba natančne metode izračuna trenutno smotrna za srednje velike vhodne podatke s 150 taksonomskimi enotami in do 50 drevesi, dejanske številke pa so odvisne od lastnosti vhodnih dreves. V primeru težav lahko posežemo po aproksimacijskih metodah, kjer na težave s časom izvajanja nebi smeli naleteti.

Št. taks. enot	Št. dreves	AMT [s]	Bi-AMT [s]	MM-AMT [s]	AMT (PMC) [s]
20	10	41.457	8.177	3.141,	2.34
20	12	81.37	11.338	5.818	3.003
20	14	212.759	15.901	7.954	3.893
20	16	366.422	19.429	13.749	4.743
20	18	735.516	25.153	17.771	6.054
20	20	1455.173	29.329	16.85	7.192
20	22	2559.055	35.694	22.479	7.565
20	24	4896.345	42.86	27.536	8.779
10	10	0.548	1.445	0.4108	0.355
12	10	1.074	2.252	0.659	0.545
14	10	2.062	3.252	1.045	0.854
16	10	3.906	4.356	1.874	1.134
18	10	13.484	5.629	2.2542	1.585
20	10	34.7066	7.288	2.4592	2.032
22	10	80.703	9.043	4.1872	2.572
24	10	113.285	11.187	5.2434	3.295
10	20	2.113	6.574	1.442	1.041
12	20	3.412	10.151	3.235	1.822
14	20	15.640	14.600	6.235	2.901
16	20	65.634	19.856	9.570	4.232
18	20	274.018	26.216	14.493	5.368
20	20	34.7066	30.995	17.016	6.411
22	20	1319.864	37.136	23.785	8.336
24	20	3173.812	45.432	25.096	10.260
50	10	-	51.875	17.257	16.557
50	12	-	77.699	75.375	22.452
50	14	-	106.556	118.352	30.098
50	16	-	140.845	165.747	39.107
50	18	-	193.063	252.198	53.963
50	20	-	234.036	314.694	65.845
50	22	-	268.463	371.333	81.322
50	24	-	320.893	453.626	129.126
50	26	-	378.354	595.073	129.126
50	28	-	439.304	747.025	319.833
50	30	-	505.951	927.528	739.345
50	32	-	576.31	1114.376	1543.093
50	34	-	645.584	1235.684	2851.712

Tabela 5.5: Časi izvajanja vseh treh metod in natančne metode s pomočjo programa PMC.

Poglavje 6

Zaključek

Najpomembnejši rezultat dela je zagotovo prva znana implementacija konsenzne metode asimetričnega srednjega drevesa v programski paket Biopython. Čeprav je računska kompleksnost metode v splošnem eksponentna (sliki 5.7 in 5.8), kar je nezaželeno, in se problemi z izvajalnim časom za natančno metodo pojavijo hitro, ima uporabnik možnost namestitve zunanjega programa PMC, ki bistveno pospeši izvajanje. Kljub temu je za velike vhodne množice dreves izračun lahko dolgotrajen, čeprav to ne velja v splošnem, saj lastnosti vhodnih dreves (število kompatibilnih poddreves) igrajo velik faktor pri iskanju največje neodvisne množice, ki je računsko najintenzivnejši del. Če uporabnik naleti na težave s časom izvajanja, lahko izbere eno izmed aproksimacijskih metod, katerih časovna zahtevnost le manjša. Pravilna izbira aproksimacijske metode je, kot smo pokazali tudi eksperimentalno, odvisna predvsem od lastnosti dreves v vhodni množici.

Kot smo pokazali na štirih vhodnih množicah, asimetrično srednje drevo ni nujno tisto, ki drevesa vhodne množice glede na Robinson-Fouldsovo metriko povzema najbolje. Prav tako kot pri izbiri aproksimacijske metode tudi tukaj velja, da je izbira konsenzne metode odvisna predvsem od predhodnega poznavanja lastnosti dreves vhodne množice. Če ta vsebujejo veliko število nekompatibilnih parov poddreves ali veliko poddreves ni večinsko zastopanih, je izbira metode asimetričnega srednjega drevesa smotrna, v nasprotnem primeru pa lahko že večinsko konsenzno drevo da primerljive, če ne celo boljše rezultate.

Kar se tiče razrešenosti dreves, smo na realnem in nekaj umetnih primerih pokazali, da je asimetrično srednje drevo vedno bilo najbolje razrešeno in je tako ponujalo največ informacij o evolucijski zgodovini, s čimer je prekašalo vse ostale metode. Tega sicer ne moremo trditi za obe aproksimacijski metodi. Metoda z izračunom največjega ujemanja je bila po razrešenosti v nekaterih primerih primerljiva zgolj s striktnim konsenznim drevesom, v drugih pa je bila nekoliko boljša od adamovega konsenznega drevesa. Metoda z izračunom drevesa iz dveh vhodnih dreves je ob primerljivi razrešenosti glede na večinsko konsenzno drevo običajno proizvedla drevo, ki se slabše ujema z vhodnimi drevesi.

Možnosti za izboljšave vidimo predvsem pri aproksimacijskih algoritmih. Nad drevesi, izračunanimi s pomočjo prvega aproksimacijskega algoritma, bi npr. lahko ponovno izračunali konsenzna drevesa in izbrali najboljšega. Poleg tega bi lahko izračun asimetričnega srednjega drevesa za vsak par vhodnih dreves opravili paralelno.

Eksponentni čas za konstrukcijo asimetričnega srednjega drevesa bi lahko zmanjšali z uporabo katerega izmed aproksimacijskih algoritmov za iskanje največje neodvisne množice. Prav tako bi bilo zanimivo določiti lastnosti množice vhodnih dreves, za katero bi metoda asimetričnega srednjega drevesa data boljše rezultate od ostalih konsenznih metod.

Literatura

[1] C. Darwin. "Notebook B: Transmutation of species", str. 36, 1837-1838.

[2] C. Phillips, T. J. Warnow. "The asymmetric median tree - A new model for building consensus trees", *Discrete Applied Mathematics*, št. 71, str. 311–335, 1996.

[3] D. Bryant. "A classification of consensus methods for phylogenetics.", *Bioconsensus*, str. 163–185.

[4] D. Gusfeld. "Efficient algorithms for inferring evolutionary trees", *Networks*, št. 21, str. 19-28, 1991.

[5] J. Felsenstein. *Inferring Phylogenies*. Sinauer Associates, 2004.

[6] J. Felsenstein. "Evolutionary Trees from DNA Sequences: A Maximum Likelihood Approach", *Journal of Molecular Evolution*, št. 17, str. 368-376, 1981.

[7] T. Y. Berger-Wolf. "Properties of compatibility and consensus sets of phylogenetic trees". *UNM Computer Science Technical Report*, TR-CS-2004-24, 2004.

[8] T. Asano, J. Jansson, K. Sadakane, R. Uehara, G. Valiente. "Faster computation of the Robinson–Foulds distance between phylogenetic networks", *Information Sciences*, št. 197, str. 77-90, 2012.

[9] T. H. Jukes, C. R. Cantor. "Evolution of protein molecules", *Mammalian protein metabolism*, št. 3, str. 21-132, 1969.

[10] M. Kimura. “A Simple Method for Estimating Evolutionary Rates of Base Substitutions Through Comparative Studies of Nucleotide Sequences”, *Journal of Molecular Evolution*, št. 16, str. 111-120, 1980.

[11] M. A. Larkin, G. Blackshields, N. P. Brown, R. Chenna, P. A. McGettigan, H. McWilliam, F. Valentin, I. M. Wallace, A. Wilm, R. Lopez, J. D. Thompson, T. J. Gibson, D. G. Higgins. “ClustalW and ClustalX version 2”, *Bioinformatics*, št. 23, str. 2947–2948, 2007.

[12] J. Felsenstein. *PHYLIP (Phylogeny Inference Package) version 3.6.* Department of Genome Sciences, University of Washington, Seattle.

[13] S. Tavare. “Some probabilistic and statistical problems in the analysis of DNA sequences.”, *Lectures on Mathematics in the Life Sciences*, št. 17, str. 57-86, 1986.

[14] P. Lemey, M. Salemi, A. Vandamme. *The Phylogenetic Handbook: a Practical Approach to Phylogenetic Analysis and Hypothesis Testing.* Cambridge University Press, 2009.

[15] F. Ronquist, M. Teslenko, P. van der Mark, D. L. Ayres, A. Darling, S. Höhna, B. Larget, L. Liu, M. A. Suchard, J. P. Huelsenbeck. “MrBayes 3.2: Efficient Bayesian Phylogenetic Inference and Model Choice Across a Large Model Space”, *Systematic Biology*, št. 61, str. 539-542, 2012.

[16] K. Tamura, G. Stecher, D. Peterson, A. Filipski, S. Kumar. “MEGA6: Molecular Evolutionary Genetics Analysis Version 6.0”, *Molecular biology and evolution*, št. 30, str. 2725–2729, 2013.

[17] P. J. A. Cock, T. Antao, J. T. Chang, B. A. Chapman, C. J. Cox, A. Dalke, I. Friedberg, T. Hamelryck, F. Kauff, B. Wilczynwski, M. J. L. de Hoon. “Biopython: freely available Python tools for computational molecular biology and bioinformatics”, *BMC Bioinformatics*, str. 1422-1423, št. 11, 2009.

[18] E. Talevich, B. M. Invergo, P. J. A. Cock, B. A. Chapman. "Bio.Phylo: A unified toolkit for processing, analyzing and visualizing phylogenetic trees in Biopython", *BMC Bioinformatics*, št. 13, 2012.

[19] S. S. Sul, T. L. Williams. "An Experimental Analysis of Consensus Tree Algorithms for Large-Scale Tree Collections", *5th Intl. Symposium on Bioinformatics Research and Applications*, str. 100-111, 2009.

[20] R. A. Rossi, D. F. Gleich, A. H. Gebremedhin, M. M. Patwary. "A Fast Parallel Maximum Clique Algorithm for Large Sparse Graphs and Temporal Strong Components", *arXiv preprint 1302.6256*, 2013.

[21] L. A. Lewis, P. O. Lewis. "Unearthing the Molecular Phylodiversity of Desert Soil Green Algae (Chlorophyta)", *Systematic Biology*, št. 54, str. 936-947, 2005.

[22] J. E. Hopcroft, R. M. Karp. "An $n^{\frac{5}{2}}$ Algorithm for Maximum Matchings in Bipartite Graphs". *SIAM Journal on Computing*, str. 225-231, št. 4, 1973.

[23] Phylogenetics, dostopno na:
http://en.wikipedia.org/wiki/Phylogenetics

[24] Matching (Graph Theory), dostopno na:
http://en.wikipedia.org/wiki/Matching_(graph_theory)

Dodatek A

Testna programska koda

```
from Bio import Phylo
from Bio.Phylo import Consensus as cons
import dendropy
from matplotlib import pyplot

trees = list(Phylo.parse('vhodna_drevesa.tre', 'newick'))
amts = {
    'AMT': cons.amt_consensus(trees, method='no_approx'),
    'Bi-AMT': cons.amt_consensus(trees, method='bi_approx'),
    'MM-AMT': cons.amt_consensus(trees, method='maxmatch_approx')
}
other = {
    'Strict': cons.strict_consensus(trees),
    'Majority': cons.majority_consensus(trees),
    'Adam': cons.adam_consensus(trees)
}

def _plot(nx, ny, fig, i, tree, title):
    ax = fig.add_subplot(ny, nx, i)
    fig.subplots_adjust(wspace=0.5)
    Phylo.draw(tree, do_show=False, axes=ax)
    pyplot.axis('off')
```

```
    pyplot.title(title)

# Shrani vhodna drevesa v datoteke
for i, tree in enumerate(trees):
    Phylo.write(tree, 'eval/in_%d.tre' % i, 'newick')

# Shrani konsenzna drevesa v datoteke
for name, tree in amts.items() + other.items():
    tree.ladderize()
    Phylo.write(tree, 'eval/%s.tre' % name, 'newick')

# Izrisi vhodna drevesa v datoteko input_trees.pdf
input_trees = pyplot.figure(0)
x = len(trees)
for i, t in enumerate(trees):
    _plot(x, 1, input_trees, i, t, 'Vhodno drevo %d' % i)
pyplot.savefig('input_trees.pdf')

# Izrisi konsenzna drevesa v datoteko consensus_trees.pdf
output_trees = pyplot.figure(1)
for i, kt in enumerate(amts.items() + other.items()):
    k, t = kt
    _plot(3, 2, output_trees, i, t, k)
pyplot.savefig('consensus_trees.pdf')

# Nalozi drevesa z DendroPy
d_amts = {
    k: dendropy.Tree(stream=open('eval/%s.tre' % k), schema
      ↪ ='newick')
    for k in amts.keys()
}
d_other = {
    k: dendropy.Tree(stream=open('eval/%s.tre' % k), schema
      ↪ ='newick')
    for k in other.keys()
}
d_input = [
    dendropy.Tree(stream=open('eval/in_%d.tre' % i), schema
```

```
        ↪ ='newick')
    for i in range(0, len(trees))
]

tpl = '[{0:10}] Res = {1:4} RF_sum: {2:5} RFs: {3}'
for k in d_amts.keys() + d_other.keys():
    if k in d_amts:
        t = d_amts[k]
    else:
        t = d_other[k]

    diffs = []
    total_diff = 0
    edges = len(t.get_edge_set())
    t.is_rooted = False
    t.update_splits()
    for in_tree in d_input:
        if in_tree.is_rooted:
            in_tree.is_rooted = False
            in_tree.update_splits()
        d = in_tree.symmetric_difference(t)
        diffs.append(str(d))
        total_diff += d

    print tpl.format(k, edges, ','.join(diffs), total_diff)
```

Slika A.1: Testna programska koda za merjenje razrešenosti konsenznih dreves in Robinson-Fouldsove metrike. Koda naloži vhodno množico dreves in iz nje izračuna konsenzna drevesa s pomočjo šestih metod. Nato vhodna drevesa in izračunana drevesa izriše v PDF datoteki ter vsako drevo shrani v datoteko v formatu Newick. S pomočjo knjižnice DendroPy nato ta drevesa naloži in nad konsenznimi drevesi izračuna razrešenost ter Robinson-Fouldsovo metriko glede na vsako vhodno drevo.

```python
from Bio.Phylo.BaseTree import Tree
from Bio.Phylo.Consensus import amt_consensus as amt
import time

current_time = lambda: int(round(time.time() * 1000))


def _test(method, trees):
    s = current_time()
    _ = amt(trees, method=method)
    return current_time() - s

N_TAXONS = range(10, 50)
N_TREES = 10
N_ITERS = 10
METHOD = 'no_approx'

for n in N_TAXONS:
    total = 0.0
    trees = [Tree.randomized(n) for i in range(0, N_TREES)]
    for _ in range(0, N_ITERS):
        total += _test(METHOD, trees)
    print '[%s, %d taxons, %d tress]: %fms' % (METHOD, n, N_TREES, total/float(N_ITERS))
```

Slika A.2: Testna programska koda za merjenje časa izvajanja. Spremenljivka *N_TREES* določa število dreves, ki bodo naključno generirana in predstavljala vhodno množico. Nad temi drevesi nato izračunamo asimetrično srednje drevo z metodo, ki jo določa spremenljivka *METHOD*.

Dodatek B

Programska koda metode *Bio.Phylo.Consensus.amt_consensus*

```python
def amt_consensus(trees, method='no_approx'):
  """Search Asymmetric Median Tree from multiple trees

  :Parameters:
  trees: list
      list of trees to produce consensus tree

  @type trees: list of Bio.Phylo.BaseTree.Tree
  @returns: Bio.Phylo.BaseTree.Tree
  """
  import multiprocessing
  import re
  from distutils.spawn import find_executable
  from itertools import combinations
  from scipy import io
  from scipy import sparse
  from subprocess import Popen, PIPE

  species = trees[0].get_terminals()
  species_names = [s.name for s in species]
  n_species = len(species)
  n_trees = len(trees)
```

```
  cpus = multiprocessing.cpu_count()

  def _tree_encoding(tree):
    clades = [
      bs for _, bs
      in _tree_to_bitstrs(tree, species_names).items()
      if bs.count('1') != n_species
    ]
    leaves = [
      _clade_to_bitstr(cld, species_names) for cld
      in tree.find_clades(terminal=True)
    ]
    return list(set(clades + leaves))

  tree_encodings = [_tree_encoding(t) for t in trees]
  profile_bitstrings = set(reduce(
    lambda enc1, enc2: enc1 + enc2, tree_encodings
  ))
  bitstring_weights = {
    bs: len([
      i for i, t
      in enumerate(tree_encodings) if bs in t
    ]) for bs in profile_bitstrings
  }
  common_bitstrings = set(
    bs for bs, w in bitstring_weights.items()
    if w == n_trees
  )

  def _amt_val(tree_encoding):
    tree_enc_set = set(tree_encoding)
    if len(common_bitstrings - tree_enc_set) != 0 or len(
      ↪ tree_enc_set - profile_bitstrings) != 0:
      res = -float('Inf')
    else:
      res = sum([
        bitstring_weights[w]
        for w in tree_enc_set - common_bitstrings
```

```
    ])

  return res

def _incompat_graph_nx(trees):
  all_bs = reduce(lambda x, y: x+y, trees, [])
  unique_bitstrings = list(set(all_bs))
  incompatible_pairs = [
    (bs1, bs2) for bs1 in unique_bitstrings
    for bs2 in unique_bitstrings
    if not bs1.iscompatible(bs2)
  ]
  incomp_graph = nx.Graph()
  incomp_graph.add_nodes_from(unique_bitstrings)
  incomp_graph.add_edges_from(incompatible_pairs)

  return incomp_graph, unique_bitstrings

def _max_indep_set(incomp_graph, unique_bitstrings):
  def _pmc(pmc_exec_path):
    comp_bitstrings = [
      bs for bs in unique_bitstrings
      if bs not in common_bitstrings
    ]
    comp_graph = nx.complement(incomp_graph)
    comp_graph.remove_nodes_from(common_bitstrings)
    mtx = nx.to_numpy_matrix(comp_graph, nodelist=
      comp_bitstrings)
    mtx = sparse.coo_matrix(np.tril(mtx))
    io.mmwrite('/tmp/igc', mtx, field='integer')

    process = Popen([pmc_exec_path, '-f', '/tmp/igc.mtx',
      '-a', '0', '-t', str(cpus), '-r', '0'], stdout=
      PIPE)
    (output, err) = process.communicate()
    exit_code = process.wait()

    if exit_code != 0:
```

```
      raise Exception('PMC aborted... %s' % err)
    else:
     pmc_regex = re.compile('Maximum clique:\s([\d\s]*)')
     match = re.search(pmc_regex, output)
      if match is None:
        raise Exception('Couldnt find PMC output. [%s]' % output)
      elif len(match.groups()) > 2:
        pmc_out = match.group(2).split(' ')
      else:
        pmc_out = match.group(1).split(' ')
      mis_vertices = [
        comp_bitstrings[int(v)-1] for v in pmc_out
        if v != '\n'
      ] + list(common_bitstrings)
      max_val = _amt_val(mis_vertices)
    return mis_vertices, max_val

  def _netx():
    current_max_size = 0
    comp_graph = nx.complement(incomp_graph)
    comp_graph.remove_nodes_from(common_bitstrings)
    maxes = []
    for clq in nx.find_cliques(comp_graph):
      if len(clq) > current_max_size:
        maxes = [clq]
        current_max_size = len(clq)
      elif len(clq) == current_max_size:
        maxes.append(clq)
    current_max = None
    current_best_val = -float('inf')
    for clq in maxes:
      mis = clq + list(common_bitstrings)
      val = _amt_val(mis)
      if val > current_best_val:
        current_best_val = val
        current_max = mis
    return current_max, current_best_val
```

```

    pmc_exec = find_executable('pmc')
    if pmc_exec is not None:
      return _pmc(pmc_exec)
    else:
      return _netx()

  def _reconstruct(bitstrings):
    m = np.transpose(
      np.array([
        bs.to_numpy() for bs
        in sorted(bitstrings,
          reverse=True,
          key=lambda bs: int(bs)
        )
      ])
    )
    ones = np.transpose(np.nonzero(m))
    row_indices = np.unique(ones[:, 0])
    col_indices = np.unique(ones[:, 1])
    cols_by_rows_index = {
      ridx: ones[ones[:, 0] == ridx][:, 1]
      for ridx in row_indices
    }
    column_values = {cidx: [] for cidx in col_indices}

    for cell in ones:
      cols = cols_by_rows_index[cell[0]]
      valid_cols = cols[cols < cell[1]]
      max_k = np.max(valid_cols) if valid_cols.size != 0
        ↪ else None
      column_values[cell[1]].append(max_k)

    for k in column_values.keys():
      if len(set(column_values[k])) == 1:
        column_values[k] = column_values[k][0]
      else:
        raise Exception('Column %d not unique, tree does
```

```python
            not exist for given encoding.' % k)

    nodes = {
      eid: BaseTree.Clade(name=eid)
      for eid in column_values.keys() + ['root']
    }
    top_level_edges = [
      eid for eid in column_values.keys()
      if column_values[eid] is None
    ]
    other_edges = [
      eid for eid in column_values.keys()
      if column_values[eid] is not None
    ]
    # connect top level edges to root
    for eid in top_level_edges:
      nodes['root'].clades.append(nodes[eid])
    # connect other nodes between each other
    for eid in other_edges:
      nodes[column_values[eid]].clades.append(nodes[eid])
    # label terminals (tree leaves)
    max_cols_by_rows = {
      ridx: np.max(cols_by_rows_index[ridx])
      for ridx in cols_by_rows_index.keys()
    }
    for row_id in max_cols_by_rows:
      term = nodes[max_cols_by_rows[row_id]]
      term.name = species_names[row_id]

    # purge node names if they do not contain string or they contain 'root' string
    for nk in nodes.keys():
      if type(nodes[nk].name) != str or nodes[nk].name == 'root':
        nodes[nk].name = ''

    return BaseTree.Tree(root=nodes['root'])

```

```
def _amt_approx_bi(trees):
  """
  Computes AMT from input trees using approximation
  method 1 (computing AMTs from two trees at a time).
  @type trees:  list of list of _BitString
  @return: Bio.Phylo.BaseTree.Tree
  """
  best_amt = None
  best_amt_val = -float('inf')
  for comb in combinations(range(0, len(trees)), 2):
    incomp_graph, ubs = _incompat_graph_nx([
      trees[comb[0]], trees[comb[1]]
    ])
    incomp_graph.remove_nodes_from(common_bitstrings)
    verts = incomp_graph.nodes()
    top, bottom = nx.bipartite.sets(incomp_graph)
    if len(top) == 0:
      mvc = bottom
    elif len(bottom) == 0:
      mvc = top
    else:
      mates = nx.max_weight_matching(incomp_graph,
        ↪ maxcardinality=True)
      matched = mates.keys()
      top_unmatched = [
        v for v in top
        if v not in matched
      ]
      cwo = [
        adj for adj in incomp_graph[v]
        if adj in bottom
        for v in top_unmatched
      ]
      cwo += top_unmatched
      mvc = set(
        [v for v in top if v not in cwo]
        + list(set(bottom) & set(cwo))
      )
```

```
      mis = [v for v in verts if v not in mvc]
      bitstrings = list(common_bitstrings) + mis
      val = _amt_val(bitstrings)
      if val > best_amt_val:
        best_amt = bitstrings
        best_amt_val = val

    return _reconstruct(best_amt)

  def _amt_approx_mm(trees):
    """
    Computes AMT from input trees using approximation
    method 2 (maximum matching)
    @type trees: list of list of _BitString
    @return Bio.Phylo.BaseTree.Tree
    """
    incomp_graph, unique_bitstrings = _incompat_graph_nx(
      ↪ trees)
    vertices = incomp_graph.nodes()
    incomp_graph.remove_nodes_from(common_bitstrings)
    matching = nx.max_weight_matching(incomp_graph,
      ↪ maxcardinality=True)
    for match in matching.keys():
      vertices.remove(match)
    return _reconstruct(vertices)

  def _amt_noapprox(trees):
    """
    Computes AMT from input trees.
    @type trees: list of list of _BitString
    @return Bio.Phylo.BaseTree.Tree
    """
    incomp_graph, unique_bitstrings = _incompat_graph_nx(
      ↪ trees)
    mis, mis_val = _max_indep_set(incomp_graph,
      ↪ unique_bitstrings)
    return _reconstruct(mis)

```

```
  if method == 'no_approx':
    amt = _amt_noapprox(tree_encodings)
  elif method == 'bi_approx':
    amt = _amt_approx_bi(tree_encodings)
  elif method == 'maxmatch_approx':
    amt = _amt_approx_mm(tree_encodings)
  else:
    raise Exception('Invalid method name. Given: %s; available: no_approx, bi_approx, maxmatch_approx' % method)

  return amt
```