Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Vizualizácia evolučných histórií

Bakalárska práca

2016
Dávid Simeunovič

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Vizualizácia evolučných histórií

Bakalárska práca

Študijný program:	Informatika
Študijný odbor:	2508 Informatika
Školiace pracovisko:	Katedra informatiky
Školiteľ:	doc. Mgr. Bronislava Brejová, PhD.

Bratislava, 2016
Dávid Simeunovič

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Dávid Simeunovič
Študijný program: informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Vizualizácia evolučných histórií
Visualization of evolutionary histories

Cieľ: Počas evolúcie dochádza v DNA k lokálnym mutáciám, ktoré menia jeden alebo niekoľko susedných nukleotidov, ale aj k väčším zmenám, ktoré menia poradie alebo počet výskytov dlhších oblastí. Cieľom práce je implementovať systém na vizualizáciu evolučnej histórie jednej alebo viacerých DNA sekvencií s dôrazom na tieto väčšie zmeny. Samotná história je daná na vstupe a cieľom je zobraziť ju tak, aby sa dali prehľadne sledovať jednotlivé mutácie a tiež vzťahy rôznych častí sekvencie.

Vedúci: doc. Mgr. Bronislava Brejová, PhD.
Katedra: FMFI.KI - Katedra informatiky
Vedúci katedry: doc. RNDr. Daniel Olejár, PhD.

Dátum zadania: 28.10.2014

Dátum schválenia: 28.10.2014 doc. RNDr. Daniel Olejár, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Poďakovanie:

Abstrakt

Počas evolúcie dochádza v DNA k lokálnym mutáciám, ktoré menia jeden alebo niekoľko susedných nukleotidov, ale aj k väčším zmenám, ktoré menia poradie alebo počet výskytov dlhších oblastí. Cieľom práce je implementovať systém na vizualizáciu evolučnej histórie jednej alebo viacerých DNA sekvencií s dôrazom na tieto väčšie zmeny. Samotná história je daná na vstupe a cieľom je zobraziť ju tak, aby sa dali prehľadne sledovať jednotlivé mutácie a tiež vzťahy rôznych častí sekvencie.

Kľúčové slová: vizualizácia,evolučná história,poradie génov

Abstract

Abstract in the English language (translation of the abstract in the Slovak language).

Keywords:

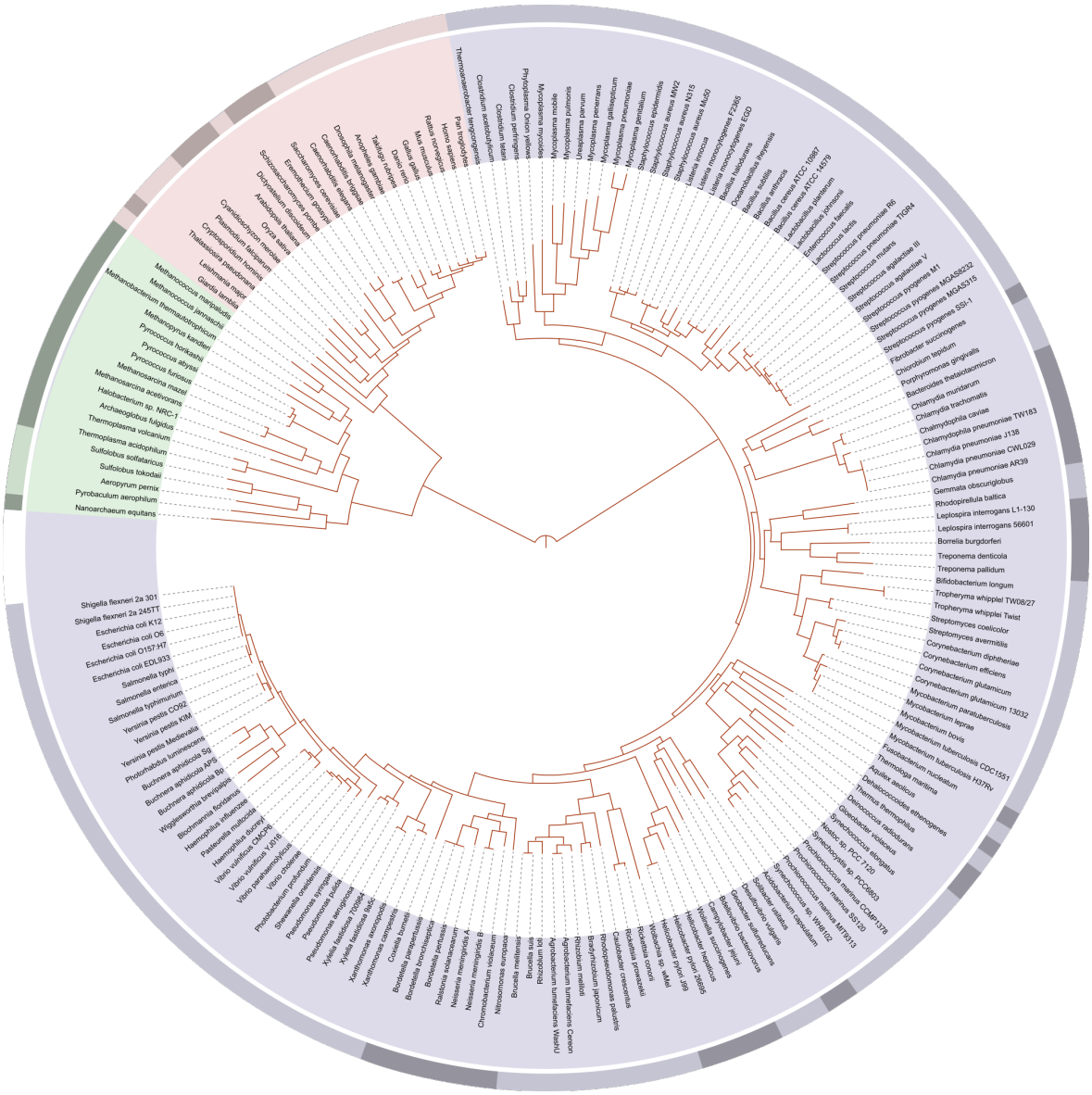
Obsah

Zoznam obrázkov

Úvod

Už v antike sa niektorý antický folozofi zaoberali myšlienkou, že základ jendotlivých druhov sa časom mení. A práve v gréčtine môžme nájsť pôvod slova *Fylogenéza*, kde fylé = kmeň a genesis = zrodenie/pôvod. Fylogenéza sa zaoberá vývojom druhu organizmov, väčšinov sa odohráva počas príliš dlhého časového úseku, preto ju niesme schopný priamo pozorovať a nezostáva nám iná možnosť ako vytvoriť rekonštrukciu na základe evolučných poznatkov. Neskôr v tomto vednom poli urobil významný pokrok Charles Darwin keď v roku 1859 publikoval svoju knihu *Pôvod druhov*, okrem toho, že z evolúcie vytvoril široko uznávanú teóriu, predstavil aj myšlienku spoločného predka, kedy akékoľvek dva veľmi rozdielne druhy zdieľajú spoločného prapredka a vizuálne ju znázornil vo forme stromového grafu, takzvaného *stromu života*. Tak položil základy Evolučnej histórie, ktorá skúma evolučné procesy, ktoré na zemi vytvorili rôznorodosť života z počiatočnej živej formy. Ďalšie poznatky v oblasti genetiky, súvisejúce s DNA a RNA viedli k tomu, že na evolúciu sa dodnes pozeráme hlavne prostredníctvom génov. Sekvenovanie DNA umožinilo vzťahy medzi jednotlivými organizmami odsledovať na základe zmien ktoré prebehli v ich DNA sekvencie. To nám ponúka množstvo presných dát využitelných pri rekonštrukcii fylogenetického procesu. Aj na našej fakulte vzniklo niekoľko prác, ktoré sa venujú rekonštrukcii DNA sekvencie pokiaľ poznáme jej súčasný vzhľad, prípadne sa snažia zrekonštruovať fylogenetický strom pokiaľ poznáme DNA sekvencie súčasných druhov. [4, 2, 1, 7].

Strom stále patrí medzi najpopulárnejšie spôsoby ako zobraziť evolučné vzťahy medzi druhmi alebo inými objektami. Najčastejšie sa stretávame so stromom života, kedy je snaha zobraziť vývoj druhov z posledného spoločného prapredka, ako napríklad vidíme na obrázku 1. V týchto prípadoch sa samotné zmeny DNA do zobrazenia nezvyknú dostať, informácie ktoré nám poskytnú, ako napríklad vzdialenosť dvoch objektov na základe rozdielnosti ich DNA sekvencie, však bývajú použité na zostavenie takéhoto zobrazenia. Cieľom tejto práce je zostrojiť program ktorý nám zobrazí jednoduchú postupnosť sekvencí DNA s dôrazom na zmeny ktoré sa udiali s génmy v týchto sekvenciách. Výsledok by mal predstavovať malú vetvu fylogenetického stromu v ktorom prepojenie objektov zobrazí reálne zmeny ktoré sa odohrali na ich DNA sekvencii. Inšpiráciou pre túto prácu sú vyššie spomenuté práce pochádzajúce z našej fakulty, náš program má byť schopný vizualizovať výsledky ktoré produkujú a poslúžiť okrem



Obr. 1: Strom života, Zdroj: wikipedia.org

iného ako rýchla optická kontrola správnosti.

Prvá kapitola nám poskytne úvod do problemtaiky, predstavíme si základné pojmy potrebné pre našu prácu.

Druhá kapitola sa bude venovať implementácii nášeho programu, pozrieme sa na to ako vyzerá jeho vstup a výstup, aké možnosti interakcie poskytuje uživateľovi a ktoré nastavenia v ňom vieme meniť.

Tretia kapitola nám povie prečo chceme zobraziť iba niektoré gény a akým spôsobom ich budeme vyberať.

Štvrtá kapitola nadväzuje na tretiu, porovnávame v nej aké výsledky dostaneme v závislosti od toho, aký spôsob výberu génov zvolíme.

Kapitola 1

Úvod do problematiky

V tejto kapitole si vysvetlíme základne pojmy potrebné pre túto bakalársku prácu.

1.1 Biologické pozadie

1.1.1 DNA,Gén,Genóm

DNA Deoxyribonukleová kyselina je nositeľom genetickej informácie bunky.
Má štruktúru dvojzávitnice, skladajúcej sa z dvoch komplementárnych vlákien. Vlákno je tvorené nukleotidmy, ktoré obsahujú jednu zo štyroch báz Adenín,Guanín,Tymín a Cytozín. DNA zvykneme zapisovať ako postupnosť týchto báz, kde každú bázu kódujeme jej počiatočným písmenom - A,G,T,C.

Gén Gén je súvislý úsek DNA ktorý kóduje tvorbu proteínu. Gén je základnou jednotkou dedičnosti.

Genóm Genóm je súbor DNA molekúl organizmu, ktoré sa väčšinou nachádzajú v chromozómoch. Napríklad v ľudskom tele sa jedná o 46 molekúl DNA, jedna v každom chromozóme[6].

1.1.2 Evolučná história

Evolučná história je postupnosť udalostí, ktoré sa odohrali na nejakej DNA sekvencii. Pre potreby tejto práce sú podstatné udalosti odohrávajúce sa na dlhých úsekoch DNA - génoch.
Možné udalosti sú:

- *Duplikácia* - skopírovanie génu na iné miesto v DNA.

- *Inzercia* - vloženie nového génu.
- *Delécia* - odstránenie génu.
- *Inverzia* - zmena poradia a orientácie génu alebo génov.
- *Translokácia* - zmena poradia génu alebo génov
- *Speciácia* - špeciálna udalosť, ktorá označuje vznik nového druhu. Vzniká nová vetva v evolučnej histórii.

Krok evolučnej histórie

Krok evolučnej histórie, ďalej len *krok e.h* je pre nás známa sekvencia génov. Medzi jednotlivími krokmi došlo k jednej alebo viacerím udalostiam.

1.1.3 Fylogenetický strom

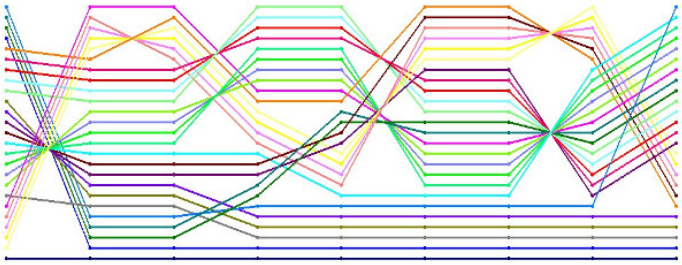
Fylogenetický strom je grafické znázornenie, ktoré zobrazuje evolučné vzťahy medzi sadou objektov. Pokiaľ si za objekty zvolíme druhy, jedná sa o takzvaný *Druhový strom*. Jednotlivé druhy sú pospájané hranami, ktoré reprezentujú evolučný vzťah.
Druhy, ktoré sa nachádzajú na *listoch* stromu sú buď existujúce druhy, z ktorých sa nevyvinuli nové druhy, alebo vyhynuté druhy bez potomkov.
Vnútorné vrcholy predstavujú predchodcov, o ktorých sa predpokladá že sa vyskytli počas evolúcie.
Pokiaľ je v strome známy posledný spoločný predok, nazveme ho *koreň*, a takýto strom označíme ako *zakorenený*.
V *zakorenenom* strome je zrejmá orientácia vnútorných hrán, ktorá určuje ktorý druh sa vyvinul z ktorého.

1.2 Vizualizácia

Vizualizácia je spôsob prevodu dát do grafickej formy, ktorú vieme spracovať naším zrakom, najdominantnejším zmyslom aký máme. To nám umožnuje okrem lepšieho pochopenia problému aj rýchlu analýzu a odhalenie existujúcich súvislostí a vzorov ktoré sa nachádzajú vo výsledku.

1.2.1 Iné programy

Iné programy použitelné pre vizualizáciu fylogenetických stromov, ako napríklad *phylo.io*,*ETE toolkit* alebo *Archaeopteryx* neponúkajú funkcionalitu ktorú potrebujeme. Poskytujú vizualizáciu fylogenetických stromov v ktorých gény buď vôbec nevystupujú alebo sa



Obr. 1.1: Výstup z programu Michaely Sandalovej, Zdroj: [6]

nachádzajú iba pri listoch stromu, nebýva na nich však zobrazený vzťah medzi jednotlivými vrcholmi stromu. Rozdiely zviknú byť vyjadrené číselne ako vzdialenosť genómov. Najväčší dôraz sa kladie na listy stromu, vrcholy nachádzajúce sa vo vnútri stromu slúžia len ako miesta pre rozvetvenie. V našej práci chceme zmeny medzi vrcholmy stromu zobraziť práve pomocou udalostí ktoré sa odohrali. Naša práca čerpá inšpiráciu z programu Michaely Sandalovej, ktorý zobrazoval preusporiadania postupnosti génov - obrázok 1.1.

Kapitola 2

Implementácia programu

V tejto kapitole sa budeme venovať niektorým významnejším črtám implementácie nášho programu, ktorý na vstupe dostane súbor popisujúci evolučnú históriu, umožní uživateľovi zmeniť niektoré nastavenia a prípadne spustiť optimalizáciu a nakoniec zobrazí grafickú reprezentáciu vstupnej evolučnej histórie podľa podľa toho aké zmeny vykonal uživateľ. Bližšie sa pozrieme na to v akom formáte má byť zapísaný vstup, ako bude vyzerať výstup, aké kroky vykonájú triedy nášho programu, akým spôsobom dokáže uživateľ interagovať s programom, ako aj to ktoré nastavenia vieme meniť a čo reprezentujú.

2.1 Vstup

Vstup musí obsahovať dáta, ktoré nám umožnia vykresliť fylogenetický strom, a zobraziť v ňom k akým zmenám v genóme došlo, a ktoré udalosti sú za to zodpovedné. To znamená že v infromáciách o jednotlivých vrcholov sa budú nachádzať aj poznatky o tom, ako vyzerá genóm daného vrcholu, a aké sú vzťahy medzi týmto a genómom jeho predchodcu.

2.1.1 Formát

Vstupný súbor nášho programu bude tvoriť postupnosťou riadkov, podobná tej akú vidíme v tabulke 2.1.1. Každý riadok predstavuje jeden *krok e.h.*. Prvý riadok je koreňom danej histórie, opisuje prvotný stav sekvencie a má priradenú špeciálnu udalosť "root". Každý ďalší riadok, opisuje niektorý z nasledujúcich krokov e.h. obsahuje zoznam génov nachádzajúcich sa v tomto kroku a spolu so svojím predchodcom nám umožnuje určiť aké udalosti z 1.1.2 viedli k súčasnému stavu. Predchodca sa v súbore musí nachádzať vždy skôr než nasledovník, aj preto je prvým riadkom koreň.
Riadok obsahuje niekoľko reťazcov a čísel, oddelených medzerou alebo viacerými medzerami, ak je to potrebné pre lepšiu prehľadnosť.

predok	e1	root	0	root	1 2 1 5 4 3 2	#	-1 -1 -1 -1 -1 -1 -1
predok	e2	e1	0.05	dup	1 2 1 2 5 4 3 2	#	0 1 2 1 3 4 5 6
clovek	e3	e2	0.12	sp	1 2 1 2 5 4 3 2	#	0 1 2 3 4 5 6 7
clovek	e4	e3	0.13	del	1 2 1 2 4 3 2	#	0 1 2 3 5 6 7
clovek	e5	e4	0.14	ins	1 2 1 6 7 2 4 3 2	#	0 1 2 -1 -1 3 4 5 6
clovek	e6	e5	0.2	inv	1 -1 -2 6 7 2 4 3 2	#	0 2 1 3 4 5 6 7 8
clovek	e7	e6	0.25	leaf	1 -1 -2 6 7 2 4 3 2	#	0 1 2 3 4 5 6 7 8
simpanz	e8	e2	0.12	sp	1 2 1 2 5 4 3 2	#	0 1 2 3 4 5 6 7
simpanz	e9	e8	0.2	leaf	1 2 1 2 5 4 3 2	#	0 1 2 3 4 5 6 7

Tabuľka 2.1: Ukážka vymysleného vstupu v súčasnom formáte [6]

Význam stĺpcov:

Prvý stĺpec je názov objektu, ktorého sa týka daný riadok.

Druhý stĺpec je id riadku.

Tretí stĺpec je id predchodcu, prvý riadok má špeciálne id predchodcu s hodnotou root.

Štvrtý stĺpec je čas, v ktorom sa daná udalosť odohrala. Koreň sa nachádza v čase 0, a čas je rastúci.

Piaty stĺpec je skratka niektorej z udalostí, popísaných v sekcii 1.1.2 pokiaľ sa v danom *kroku e.h.* odohrala iba jedna z týchto udalostí, alebo jedna zo trojice udalostí root/leaf/other. Root je udalosť slúžiaca na identifikáciu koreňa. Leaf slúži na určenie času v ktorom sa daná vetva končí, medzi udalosťou označenou ako leaf a jej predchodcom nemuselo prísť k žiadnym zmenám. Other použijeme ak tento rozdiely medzi týmto a predchádzajúcim krokom niesme schopný popísať pomocou jednej udalosti, znamená to že takýto krok vznikol kombináciou viacerých udalostí ako napríklad dve duplikácie nasledujúce po sebe alebo translokácia s následnou deléciou.

Nasledujúce stĺpce obsahujú postupnosť génov. Každý gén je celé číslo, pričom znamienko určuje jeho orientáciu. To znamená že gén 2 je rovnaký ako gén -2, iba opačne orientovaný v rámci dna.

Znak # slúži ako ukončenie zoznamu génov.

Zvyšné stĺpce pre každý gén určujú poradie predka génu v predchodcovi jeho riadku. Ak tento gén nemá predchodcu, obsahuje riadok hodnotu -1. Napríklad pre gén 2

z riadku e4 vieme, že poradie jeho predchodcu má index 3. Keďže predchodcom e4 je e3 vieme spojiť štvrtý (indexujeme od nuly) gén z riadku e3 s štvrtým génom (gén 2) riadku e4.

predok	e1	root	0	root	1 2 1 5 4 3 2	#	-1 -1 -1 -1 -1 -1 -1
predok	e2	e1	0.05	dup	1 2 1 2 5 4 3 2	#	0 1 2 1 3 4 5 6
clovek	e3	e2	0.12	sp	1 2 1 2 5 4 3 2	#	0 1 2 3 4 5 6 7
clovek	e4	e3	0.13	del	1 2 1 2 4 3 2	#	0 1 2 3 5 6 7
clovek	e5	e4	0.14	ins	1 2 1 6 7 2 4 3 2	#	0 1 2 -1 -1 3 4 5 6
clovek	e6	e5	0.2	inv	1 -1 -2 6 7 2 4 3 2	#	0 2 1 3 4 5 6 7 8
clovek	e7	e6	0.25	leaf	1 -1 -2 6 7 2 4 3 2	#	0 1 2 3 4 5 6 7 8
simpanz	e8	e2	0.12	sp	1 2 1 2 5 4 3 2	#	0 1 2 3 4 5 6 7
simpanz	e9	e8	0.2	leaf	1 2 1 2 5 4 3 2	#	0 1 2 3 4 5 6 7

Tabuľka 2.2: Ukážka vymysleného vstupu v súčasnom formáte [6]

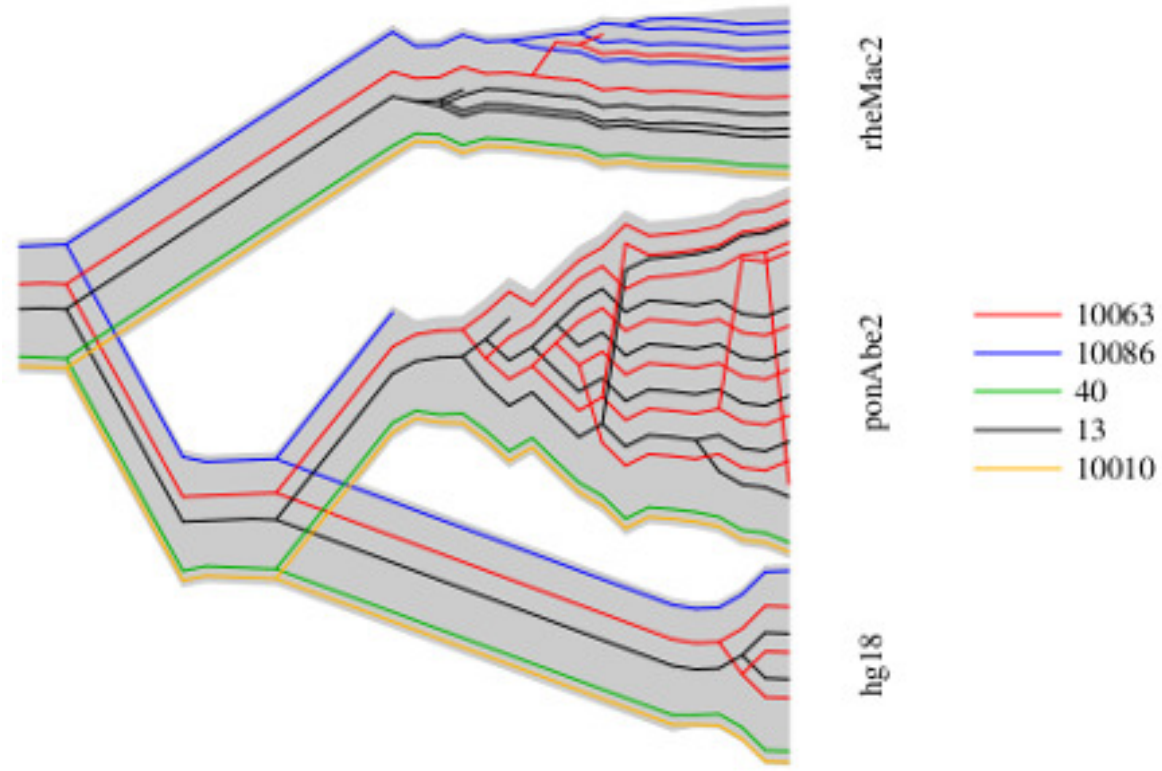
2.2 Návrh výstupu

Výstupom nášho programu by nemal byť iba jednoduchý fylogenetický strom, keďže naším cielom je zobraziť kompletnú informáciu o génoch ktoré sa nachádzajú v evolučnej histórii. Potrebujeme preto nájsť spôsob akým túto informáciu pridať do fylogenetického stormu. Rozhodli sme sa teda, že každý gén bude zobrazený počas celej jeho existencie v evolučnej histórii. Naše zobrazenie evolučnej histórie bude perdstavovať les stromov, v ktorom jednotlivé stromy reprezentujú gény nachádzajúce sa v týchto druhoch.

X-ová os slúži ako os času, naľavo sa nachádza čas 0, ktorý postupne rastie tak, aby sa do obrázku zmestil aj najneskorší krok evolučnej histórie.

Gény sú znázornené farebnými čiarami, ktoré idú vodorovne až kým sa nedostanú do bodu v ktorom má nastať udalosť. Ak vieme že nasledujúci krok k evolučnej histórie sa nachádza v čase t_k, tak gény dojdú bez zmeny do bodu $t_k - d$, kde d reprezentuje čas potrebný pre odohratie zmien. Zmeny vedúce ku kroku k sa teda udejú počas časového rozmedzia $t_k - d$ až t_k. Pozícia génov v rámci Ypsilonovej osi je v udalosti root daná ich reálnym poradím, prvý gén udalosti root sa nachádza najvyššie na Y osi. Pre každý ďalší krok platí, že gény daného kroku sú na Y osi zoradené podľa poradia v akom sa nachádzajú v tomto kroku evolučnej histórie.

Duplikáciu, teda skopírovanie jedného alebo viacerých génov zobrazíme tak, že každý zo stromov reprezentujúci zduplikované gény rozvetvíme. Vetvenie začne v čase $t_k - d$.



Obr. 2.1: Možné zobrazenie histórie, Zdroj:[7]

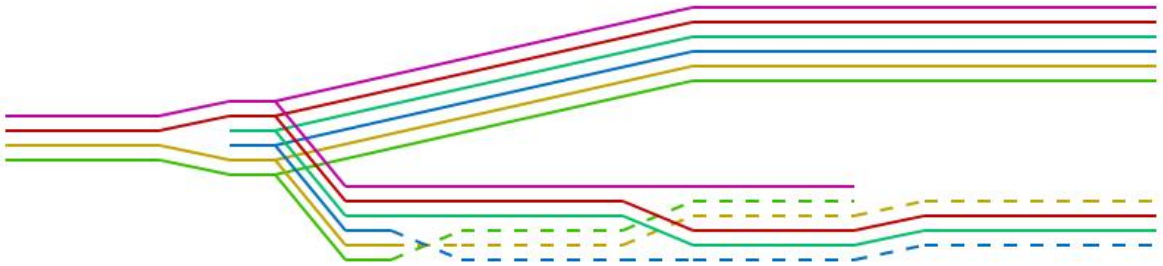
Speciáciu by sme v klasickom druhovom strome zobrazili ako rozvetvenie druhového stromu, počas speciácie teda rozvetvíme všetky gény tak ako by sa nachádzli v rozvetvení druhového stromu - získame dve sady vetiev reprezentujúce naše gény. Rozdiel medzi speciáciou a duplikáciou ktorá prebehla na celom genóme je zreteľný vo vzdialenosťi na Y osi, kedy pri speciácii rozoznávame dve oddelené sady vetiev, ktoré sú od seba dostatočne vzdialené, zatiaľ čo duplikácia by vytvorila iba jednu sadu vetiev. Ak sa jedna vetva speciácie nachádza v čase t_a a druhá v čase t_b zobrazíme začiatok speciácie do času $t_x - d$ kde t_x je skorší z časov t_a, t_a.

Inzerciu génov znázorneníme pridaním nového stromu pre každý vložený gén, začiatok pridaních stromov sa nachádza v čase t_k.

Deléciu génu zobrazíme ako ukončenie vetvy stromu, ktorá reprezentovala inštanciu tohto génu, takúto vetvu ukočíme v čase $t_k - d$.

Translokáciu zobrazíme ako kríženie vetiev translokovaných génov tak, aby sa po tomto krížení nachádzali vetvy v správnom poradí, kríženie sa začne v čase $t_k - d$ a skončí v čase t_k.

Inverziu znázorníme podobne ako translokáciu, avšak keďže pri inverzii okrem zmeny poradia génov dochádza aj k zmene ich orientácie, pridáme do zobrazenia údaj o orientácii génu. Gén so zmenenou orientáciou nebudeme zobrazovať plnou čiarou, na miesto nej využijeme prerušovanú čiaru. To nám umožní zobraziť inverziu jedného



Obr. 2.2: Príklad zobrazenia histórie naším programom

génu, ktorá by inak nemusela byť viditeľná a inverziu dvoch génov ktorú by sme si mohli v niektrých prípadoch pomýliť s translokáciou. Zmena štýlu čiari nastáva už v čase $t_k - d$.

Root je začiatok pre všetky stromy génov, ktoré sa nachádzajú v počiatočnom predkovi.

Leaf ukončí vetvi génov v čase t_k.

Obrázok 2.2 ilustruje všetky udalosti ktoré sme si práve opísali, naľavo sa nachádza krok evolučnej histórie "rootöbsahujúci štyri gény, nasleduje inzercia dvoch ďalších génov a po nej speciácia. Pri speciácii vzniknú dve vetvy génov ktoré sú od seba dostatočne vzdialené na to aby sme ich rozoznali. Navyše vrchná vetva speciácie sa začína v neskoršom čase, potom už neobsahuje žiadne udalosti iba končím v liste. V spodnej vetve sa odohráva inverzia, po nej translokácia, nakoniec delécia dvoch génov a ukončenie zvyšných v liste.

2.3 Triedy programu

V stručnosti si predstavíme základne funkcie ktoré plnia triedy v našom programe bez toho, aby sme zachádzali príliš hlboko do implementačných detailov.

Bak je hlavnou triedou ktorá sa nachádza v našom programe. Načíta vstupné parametre, a na ich záklda sa rozhodne či program prebehne neinteraktívne, iba na základe prvotného vstupu, alebo sa spustí interaktívna grafická aplikáciá využívajúcá JavaFX, ktorá je zapísaná v tejto triede.

EvolutionTree je kľúčovou triedou celého projektu, reprezentuje evolučnú históriu, obsahuje výpočty potrebné pre jej vykreslenie ako aj optimalizáciu. Obsahuje podtriedu *EvolutionNode* ktorá popisuje jeden krok evolučnej histórie. Táto trieda dostane vstupnú históriu a tú si uloží v potrebnom formáte. Pre každý krok evolučnej histó-

rie si vytvorí jeden *EvolutionNode*, krok typu *root* si priamo uloží a zvyšné kroky si namapuje podľa ich *id*.

Pri vykreslovaní zisťuje, akú šírku bude krok zaberať na obrázku. Pre list je táto šírka daná súčtom šírok jeho génov a medzier medzi nimi. Pre zvyšné kroky je daná buď ako ich vlastná šírka, to znamená výpočet rovnaký ako pri liste, alebo ako šírka kroku ktorý po ňom nasleduje, vyberáme väčšiu z hodnôt. Pokiaľ v kroku nastáva vetvenie druhového stromu, vyberieme ako jeho šírku väčšiu z hodnôt jeho vlastnej šírky, alebo súčtu šírok krokov ktoré z neho vychádzajú, a medzery, ktorú medzi nimi musíme nechať. Keď pozná šírky ktoré potrebujú jednotlivé kroky môže vykresliť našu evolučnú históriu.

V tejto triede taktiež prebieha aj hľadanie blokov 3.1.1 a následné riešenie problému množinového pokrytia greedy algoritmom alebo jeho export vo forme celočíselného lineárneho programu. Bloky hľadáme pre každú dvojicu predchodca kroku,krok zvlášť. Pokiaľ je nejaký krok predchodcom pre dva kroky, to znamená že z neho vychádzajú dve speciácie, hľadáme bloky nezávisle pre oboch jeho nasledovníkov.

DrawFactory je abstraktná trieda slúžiaca pre vykreslovanie evolučnej histórie. V našom programe sa nachádza dve triedy ktoré ju rozširujú. *FXDrawFactory* slúži na vykreslenie evolučnej histórie v našej JavaFX aplikácii. *SVGDrawFactory* slúži na vykreslenie evolučnej histórie vo formáte *svg* - škálovateľná vektorová grafika, históriu v tomto formáte vieme následne vyexportovať. *EvolutionTree* dostane inštanciu jednej z týchto dvoch tried a pomocou nej vykreslí evolučnú históriu.

Gene Meta reprezentuje nastavenia jedného génu, spôsob ako ich exportovať do XML a tiež možnosť načítať takéto nastavenia z XML súboru.

Settings V tejto triede sa nachádzajú globálne nastavenia ktoré využívame pre vykreslovanie, sú tu uložené aj *GeneMeta* pre niektoré gény, pokiaľ pre gén neexistujú *GeneMeta*, alebo neobsahujú hodnotu pre požadované nastavenie, využije sa všeobecná hodnota tohto nastavenia ktorá sa nachádza v triede *Settings*. Trieda *Settings* taktiež obsahuje omožnosť vyexportovať nastavenia, vrátane vrátane všetkých *GeneMeta*, do XML, ako aj možnosť nastavenia z XML súboru načítať.

2.4 Ovládanie programu

Náš program ponúka dve možnosti ako ho ovládať. Prvou z nich je grafická aplikácia ktorá uživateľovi umožnuje interagovať s nastaveniami a spúšťať funkcie ktoré náš program obsahuje. Druhou možnosťou je vložiť nášmu programu všetky požiadavky, ktoré má splniť v príkazovom riadku už pri jeho spustení, tento spôsob nevyžaduje

Obr. 2.3: Vzhľad grafickej aplikácie

žiadnu ďalšiu interakciu a je ideálny pri potrebe spracovať väčšie množstvo vstupov. Teraz sa pozrieme na to aké možsnosti ovládania tieto dve alternatívy ponúnkajú.

2.4.1 Grafická aplikácia

Na obrázku 2.3 vidíme základné okno našej aplikácie. V strede sa nachádza plocha na ktorú sa vykresľuje evolučná história. Na ľavo sa nachádza základná sada tlačidiel a na pravo ovládacie prvky pre zmenu nastavení. Štandardne má aplikácia pri spustení rozmery 1200x800 a plocha, na ktorú sa vykresluje rozmery 800x800.

Tlačidlá na ľavej strane slúžia k základnému ovládaniu nášho programu, predstavíme si čo ktoré tlačidlo po stlačení vykoná, a aké to má využitie.

Open otvorí nové okno ktoré nám umožní zvoliť súbor ktorý cheme otvoriť, tento súbor by mal byť vo formáte popísanom v sekcii 2.1. Zvolená evolučná história sa načíta a vykreslí.

Save img otvorí nové okno ktoré nám umožní zvoliť si do akého súboru sa uloží náš súčasný obrázok, ten sa ukladá vo formáte SVG.

Import Settings v novom oknem nám umožní vybrať si súbor vo formáte XML obsahujúci nastavenia ktoré má načítať.

Export Setting umožnuje vybrať súbor do ktorého sa uložia naše nastavenia. Nastavenia sa ukladajú vo formáte XML.

Optimize zostrojí problém množinového pokrytia tak ako je to popísané v kapitole 3 a následne ho približne vyrieši pomocou greedy algoritmu. poďla toho akú hodnotu má pole *optimization* sa toto riešenie prenesie do zobrazenia.

Export ILP podobne ako tlačidlo *Optimize* zostrojí problém množinového pokrytia, ale miesto toho aby našiel jeho približné riešenie, uloží ho zapísaný ako celočíselný lineárny program, do nami zvoleného súboru. Súbor do ktorého sa uloží vyberáme už klasicky v novom okne ktoré nám otvorí.

Import ILP v novom oknem nám opäť umožní zvoliť súbor ktorý sa otvorí, tento krát má však daný súbor obsahovať riešenia celočíselného lineárneho programu. Náš program je nastavený tak aby vedel spracovať riešenia ktoré produkuje *CPLEX*, to akým spôsobom sa načítané riešenie prenesie do zobrazenia evolučnej histórie, opäť záleží na tom, aká hodnota sa nachádza v poli *optimization*.

Redraw po stlačení tohto tlačidla dojde k prekresleniu obrázku.

Ovládacie prvky na pravej strane Na ľavej strane sa nachádzajú dve sady ovládacích prvkov. Prvá sada mení nastavenia ktoré sa nachádzajú priamo v triede *Settings* a sú zdielané všetkými génmi a celým prostredím. Druhá sada mení nastavenia špecificky pre zvolený gén, je to teda spôsob akým meníme nastavenia triedy *GeneMeta* pre konkrétny gén.

Globálne

Time diff je číslo vyjadrujúce koľko času má zabrať odohranie udalosti na X-ovej osi, ak má krok evolučnej histórie k známy čas t_k tak jeho udalosť sa začne odohrávať už v čase $t_k - \textit{Time diff}$, tento čas však nesmie byť menší ako čas v ktorom sa odohral predchodca kroku k. Time diff reprezentuje rovnakú premenn=u ako znak d v sekcii 2.2.

Width určuje dĺžku X-ovej osi, teda šírku nášho obrázku, náš program každú evolučnú históriu naškáluje tak, aby ležala na celej tejto osi.

Height určuje dĺžku Y osi, teda výšku nášho obrázku, náš program na šírku neškáluje, to aká časť šírky je pokrytá závisí od šírok génov a ich medzier.

Node Gap určuje vzdialenosť medzi dvoma sadami vetiev ktoré vznikli počas speciáce. zadáva sa celé číslo.

Line Gap určuje medzeru medzi dvoma susednými génmi ktoré sa nachádzajú v jednom kroku evolučnej histórie.

Line Size šírka čiary znázornujúcej jeden gén, platí pre všetky gény pokiaľ nemajú svoje vlastné špecifické nastavenie.

Saturation a *Brightness* . Štandardne získavame farby ktoré priradíme génom z modelu HSB - hue,saturation,brightness - slovensky odtieň,sýtosť,jas. Odtieň v tomto modeli je daný v stupňoch hodnotou $0° - 360°$. Rovnomerne rozdeliť odtiene medzi našich n génov, je teda rovnaký problém ako rozdeliť kružnicu na n rovnakých častí. Sýtosť a jas sú potom pre všetky gény rovnaké, určujeme ich reálnym číslo z rozmedzia $< 0, 1 >$.

Optimization určuje akým spôsob sa prejaví optimalizácia na výslednom zobrazení evolučnej histórie.Pri hodnote 2 sa génom, ktoré sa nenachádzajú v riešení nastaví atribút $higlighted = false$, vykreslia sa farbou ktorá je daná ako *Non highlighted*, takéto nastavenie vidíme na obrázku [?] v okinenku číslo 2. Pri hodnote 1 sa nezvolením génom nastaví atribút $trasnparent = true$, zobrazia sa priesvitne, rovnako ako v okienku číslo 3. Pri hodnote 0 sa nezvolením génom nastaví atribút $draw = false$, a teda sa nezobrazia vôbec, rovnako ako štvrtom okienku. Génom ktoré sa nachádzajú v riešení sa atribúty nemenia ani v jednom z týchto nastavení.

Non highlighted umožnuje výber farby pre gény, ktoré nie sú zvýraznené. To sú tie pre ktoré $highlighted = false$ a zároveň $draw = true$ a $transparent = false$. Tieto hodnoty génom manuálne nastavíme alebo ich získajú pokiaľ sa nenachádzajú v riešení optimalizácie a $optimize = 2$.

Redraw je preínač, pokiaľ je zapnutý obrázok sa automaticky prekreslí po tom, čo zmeníme niektoré z jeho nastavení. V opačnom prípade obrázok prekreslíme stlačením ľavého tlačidla *redraw*

Špecifické pre gén sa aplikujú na vybraný gén ktorý zvolíme z rozklikávacieho listu. V tomto liste sa nachádzajú všetky gény z histórie, bez ohľadu na to, či pre ne existuje záznam *GeneMeta* uložení v nastaveniach *Settings*. Okrem toho sa v liste nachádzajú dve špecíalne hodnoty. *None* vyjadruje že nemáme zvolený žiaden gén a nebudeme vedieť meniť žiadne nastavenia špecifické pre gén. *Default* slúži k nastaveniu hodnôt, ktoré využívajú všetky gény bez vlastných *GeneMeta*. Pre *Default* nevieme nastavovať *Name* ani *Colour*. Pokiaľ si z listu zvolíme gén ktorý nemá vlastný záznam *GeneMeta*, tento záznam sa vytvorí až keď tomuto génu nastavíme niektorú z hodnôt. Do polí *Draw,Transparent a Highlighted* sa prekopíruje hodnota nastavená v *Default*, zvyšné polia môžu zostať prázdne. Pre prázdne polia sa aj naďalej využívajú hodnoty z *Default*.

Draw boolean prepínač, hodnota určuje či sa gén vykreslí.

Transparent boolean prepínač, pokiaľ sa má gén vykresliť ($draw = true$), tak hodnota *Transparent* určí, či bude priesvitný $transparent = true$, kedy gén nevidíme, ale zaberá miesto. Alebo bude nepriesvitný, kedy farbu získa v závislosti od hodnoty *Highlighted*.

Highlighted pokiaľ sa má gén nepriesvitne vykresliť $draw = true, transparent = false$ highlighted rozhodujem o tom či bude zvýraznený. Ak je zvýraznený zobrazí sa s farbou ktorú má nastavenú v *Colour*, v prípade že nemá nastavenú vlastnú farbou použije predpočítanú farbu. Ak nieje zvýraznený použije farbu nastavenú v *Non Highlighted*.

Line Width umožnuje individuálne nastavenie šírky génu. Vkladáme celé číslo.

Name je textové pole v ktorom môžme génu nastaviť jeho meno.

Colour ponúka výber farby špeciálne pre daný gén.

Delete gene meta odstráni záznam *GeneMeta* pre práve zvolený gén.Nefunguje pre *Default*.

Delete all meta odstráni všetky existujúce záznamy *GeneMeta* okrem záznamu *Default*.

2.4.2 Argumenty príkazového riadku

Nie vždy pre nás musí byť výhodnejšie, ovládať náš program cez grafickú aplikáciu. Ako alternatívu ku grafickej aplikácii ponúka náš program možnost, vložiť mu základné parametre potrebné pre jeho beh už pri jeho spustení. Program potom sám vykoná zvolené úkony, bez potreby akejkoľvek ďalšej interakcie zo strany uživateľa. Tento spôsob ovládania programu je výhodný napríklad vtedy, keď cheme automatizovať spracovanie väčšieho počtu vstupov. Predstavíme si aké argumenty vieme vložiťna príkazový riadok, a aký vplyv budú mať na priebeh nášho programu. Každý argument ktorý týtmto spôsobom programu vložíme má na začiatku pomlčku. Poradie argumentov nie je dôležité.

-nogui argument určujúci že náš program nespustí grafickú aplikáciu, ale vystačí si len s údajmi zo vstupu.

-input:filename.history argument, ktorým ukážeme na súbor "filename.history" obsahujúci evolučnú históriu. Evolučná história ktorá sa nachádza v tomto súboré sa načíta do programu.

-drawsvg nahraná evolučná história sa vykreslí vo formáte svg do súboru output.svg.

-svg_ output:filename.svg miesto súboru output.svg sa obrázok uloží do súboru "filename.svg".

-opt:x prebehne optimalizáciu počtu zobrazených génov, číslo x má rovnakú funkciu ako nastavenie *Optimization* z grafickej aplikácie.

-exportgreedy do súboru output.greedy uloží greedy riešenie problému množinového pokrytia pre nahranú evolučnú históriu.

-greedy_ output:filename.svg miesto do súboru output.greedy sa greedy riešenie uloží do súboru "filename.svg".

-exportilp zadanie celočíselného lineárneho programu - 3.3.2, ktorý je ekvivalentný k problému množinového pokrytia pre nahranú evolučnú históriu, sa uloží do súboru output.lp

ilp_ output:filename.lp namiesto output.lp sa pre uloženie zadanie celočíselného lineárneho programu využije súbor "filename.lp".

-load_ lp:input.sol optimalizuje zobrazené gény na základe riešenia zo súboru ”input.sol". Program je nastavený tak aby akceptoval riešenie ktoré prodkuje CPLEX.

-load_settings:filename.xml do triedy *Settings* načítana nastavenia zo súboru "filename.xml".

Kapitola 3

Problém množinového pokrytia a výber génov

V predchádzajúcej kapitole sme si predstavili základné prvky nášho programu ktorý dostane na vstupe súbor, popisujúci evolučnú históriu a na výstupe nám vykreslí fylogenetický strom reprezentujúci danú históriu. Problém nastane, pokiaľ v histórii nachádza príliš veľa génov. Výsledný vygenerovaný obrázok sa stáva neprehľadným, a získanie informácie z neho obtiažne. Potrebujeme teda vybrať iba niektoré gény na zobrazenie tak, aby na obrázku zostali zachované podstatné informácie. V tejto kapitole si predstavíme spôsob, akým budeme vyberať ktoré gény zobrazíme, využitie *Problému množinového pokrytia* pri hľadaní daných génov a dva algoritmy ktoré riešia daný problém.

3.1 Výber génov

Najpodstatnejšou informáciou pri analýze fylogenetického stromu je pre nás to, aké udalosti sa v ňom odohrali. Budeme sa teda snažiť nájsť podmnožinu všetkých génov tak, aby všetky udalosti ostali na obrázku zachované. Zvyšné gény následne z obrázku odstránime, čo môže viesť k strate informácií ktoré považujeme za menej podstatné, ako napríklad to, koľko a ktoré geny sa nachádzajú v danej histórii, ako aj koľko a ktoré gény sú ovplyvné danou udalosťou.

3.1.1 Blok

Blok predstavuje postupnosť génov ktoré sa pred aj po kroku e.h. nachádzali vedľa seba v rovnakom poradí a jednotlivé gény nemenili svoju orientáciu. Jedná sa teda o súvislý úsek DNA ktorý počas kroku e.h. nebol prerušený. Ak sa pri delécii alebo inzercii odobralo alebo pridalo viacero génov, a nenachádza sa medzi nimi žiaden iný gén, tvoria jeden blok. Pri duplikácii gény tvoria blok ak sa nachádzali pri sebe pred

duplikáciou a rovnako aj po nej vo všetkých zduplikovaných inštanciách. Pri Inverzii sa gény nachádzajú v bloku pokiaľ sa všetkým zmení orientácia, t.j. zrotuje celý blok. Napr blok génov (4,5,-6) bude po inverzii vyzerať ako (6,-5,-4).

Pokrytie blokov

Blok považujeme za pokrytý pokiaľ sa na obrázku vyskytuje aspoň jeden gén patriaci do daného bloku. Pokrytie všetkých blokov jedného kroku e.h nám zaručí zobrazenie všetkých udalostí, aj keď nie v úplnom rozsahu, ktoré sa v danom kroku vyskytli. Musíme preto nájsť také gény, ktoré pokryjú všetky bloky v kompletnej evolučnej histórii, a tým si zaistiť zobrazenie všetkých udalostí vo výslednom fylogenetickom strome. Ako cieľ si zvolíme aby bola daná množina génov čo najmenšia.

3.2 Problém Množinového Pokrytia

Definícia Máme dané univerzum$U$, ktoré obsahuje n prvkov, a systém jeho podmnožín $S = \{P_i : P_i \subseteq U\}$, ktorý pokrýva celé univerzum $\cup_{Pi \epsilon S} Pi = U$, vybrať čo najmenšiu množinu podmnožín $C \subseteq S$ takú ktorá tiež pokryje celé Univerzum $\cup_{Pi \epsilon C} Pi = U$. Problém Množinového Pokrytia (anglicky Set Cover Problem), ďalej len *PMP*. patrí medzi NP-úplne problémy.[3]

Príklad Pre Univerzum$U = \{1, 2, 3, 4, 5, 6\}$
a systém jeho podmnožín $S = \{\{1, 2, 3\}, \{2, 3\}, \{3, 4\}, \{3, 4, 6\}, \{5\}\}$
je riešením množina podmnožín $C = \{\{1, 2, 3\}, \{3, 4, 6\}, \{5\}\}$

3.2.1 Výber génov pomocou Problému Množinového Pokrytia

Výber takých génov ktoré pokryjú všetky bloky v celej evolučnej histórii vieme formulovať ako Problém Množinového Pokrytia Univerzum predstavuje všetky bloky ktoré sa nachádzajú v našom fylogenetickom strome. Každý gén predstavuje jednu podmnožinu, v ktorej sa nachádzajú tie bloky, cez ktoré gén prechádza. Riešením je taká množina génov, ktorých zjednotenie pokrýva všetky prvky Univerza, v našom prípade všetky bloky nachádzajúce sa v evolučnej histórii.

3.3 Riešenie Problému Množinového Pokrytia

Keďže *PMP* patrí medzi NP-ťažké problémy, znamená to že zatiaľ neexistuje, a možno nikdy ani nebude existovať algoritmus ktorý by dokázal nájsť riešenie v polynomiálnom čase. Potrebujeme sa teda rozhodnúť, či je pre nás výhodnejšie hľadať najlepšie riešenie *PMP* čo môže byť časovo náročné, alebo sa uspokojíme s približním riešením, ktoré

sme schopný nájsť aproximačným algorimtom v polynomiálnom čase, a ktoré môže taktiež predstavovať dostatočné odstránenie prebytočných génov z obrázku. Predstavíme si jeden spôsob ktorým budeme hladať úplné riešenie, jeden spôsob na nájdenie približného riešenia a v nasledujúcej kapitole porovnáme výsledky ktoré produkujú.

3.3.1 Greedy algoritmus

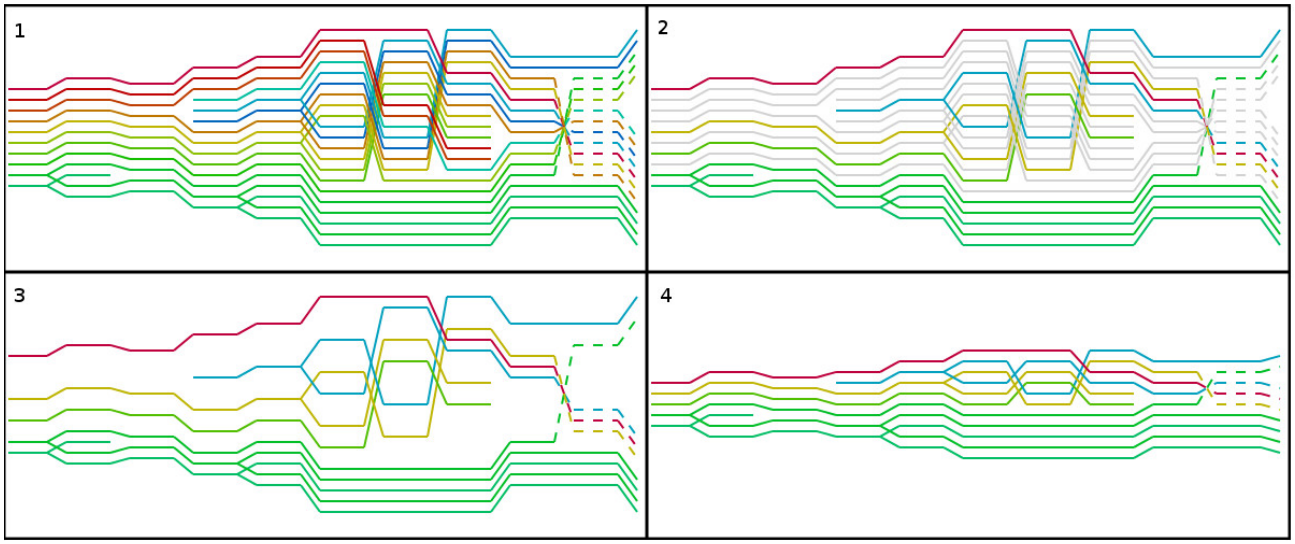
Greedy algoritmus patrí medzi najlepšie polynomiálne aproximačné algoritmy pre riešenie *PMP*.[5] Greedy algoritmus v každom kroku pridá do riešenia takú podmnožinu, ktorá obsahuje najviac zatiaľ nepokrytých prvkov univerza. Riešenie teda hľadáme nasledovným spôsobom:
Všetky podmnožiny zoradíme na základe toho, koľko prvkov obsahujú. Do riešenia vyberieme najvačšiu podmnožinu a prvky, ktoré sa v nej nachádzajú odstránime z univerza aj zo zvyšných podmnožín. Zvyšné podmnožiny opäť zoradíme podľa veľkosti, a postup opakujeme až pokým nie je Univerzum prázdne.
Tesná analýza podľa Slavíka ukazuje, že aproximačný koeficient takéhoto riešenia je $\ln m - \ln \ln m \ + \Theta(1)$ [5] kde $m = |U|$.

3.3.2 Binárne-Celočíselné Lineárne Programovanie

Lineárne programovanie je optimalizačná úloha, pri ktorej je cieľom nájsť minimum alebo maximum lineárnej funkcie f s n-premennými, zatiaľ čo máme dané lineárne obmedzenia vo forme rovníc a nerovníc. V prípade binárneho-celočíselného lineárneho programovania nadobúdajú premenné hodnotu 0 alebo 1, a všetky atribúty obmedzujúcich rovníc a nerovníc sú celočíselné. Binárne-Celočíselné lineárne programovanie (angl: 0-1/binary integer linear programming),ďalej BILP patrí medzi NP-úplné problémy.[3] Množstvo iných problémov, ako napríklad Problém obchodného cestujúceho, Problém Vrcholového pokrytia a *PMP* môžu byť formulované ako Celočíselné linárne programovanie. Navyše pre Celočíselné lineárne programovanie existuje množstvo

Prevedenie výberu génov na BCLP

Všetky gény nachádzajúce sa v našej evolučnej histórii očíslujeme číslom $1 - n$, premenná x_i bude nadobúdať hodnotu 0 alebo 1 v závislosti od toho či sa gén pod číslom i nachádza v riešení. Lineárna fukncia ktorú cheme minimalizovať bude v tvare $min\ x_1 + x_2 + x_3 + .. + x_n$. Lineárne obmedzenia vytvoríme tak, že pre každý blok B_a sa pozrieme na všetky gény ktoré daný blok pokrývajú x_{ai} : $x_{ai} \epsilon B_a$, a pridáme podmienku že súčet premmených x_{ai} reprezentujúcich takéto gény musí byť väčší ako 1, $\{x_{a1} + x_{a2} + .. + x_{aj} \geq 1 | \forall i_{i \epsilon \{1..j\}} : x_{ai} \epsilon B_a\}$, to znamená že v riešení sa musí nachádzať aspoň jeden gén pokrývajúci daný blok. Vo výslednom *CLP* sa bude nachádzať



Obr. 3.1: Kroky optimalizácie

jedna lineárna funkcia ktorú chceme minimalizovať, tá obsahuje n premenných kde $n =$ počet génov $= |S|$. Plus k lineárnych obmedzení, kde $k =$ počet blokov $= |U|$.

Riešenie BCLP Cieľom tejto práce nie je nájsť najlepší spôsob, alebo zostrojiť najlepší program, pre riešenie *BCLP* či *PMP*. Oba problémy sú len prostriedkom ako dosiahnuť optimalizáciu množstva zobrazených génov a tým zvýši prehladnosť. V prípade greedy algoritmu sa implementácia nachádza priamo v našom programe, čo nám umožnuje v krátkom čas dospieť aspoň k čiastkovej optimalizácii. V prípade BCLP náš program nevie nájsť riešenie, ponúka však možnosť vyexportovať sformulovaný *BCLP*. Pre samotné riešenie *BCLP* je vhodné použit niektorý z existujúcich nástrojov[8], a riešenie nahrať do nášho programu. pre účely tejto práce bol využívaný IBM ILOG CPLEX Optimization Studio - *CPLEX*

3.3.3 Výsledok optimalizácie

Výsledkom optimalizácie génov si môžme ilustrovať na obrázku 3.1. Na začiatku dostane náš program evolučnú históriu s kompletnou informáciou, ako vidíme v časti 1). Následne zostrojíme všetky bloky a nájdeme riešenie pre daný *PMP*. V časti 2) sú gény, ktoré sa nachádzajú v riešení vyznačené farebne, zvyšné gény sú šedé. Prebytočné gény z obrázku odstránime. V časti 3) môžme pozorovať, ako sa ich odstránením obrázok preriedi, napriek tomu ostávajú všetky udalosti zachované. Časť 4) je finálnym krokom optimalizácie, prebytočné gény už viac nezaberajú žiadne miesto, všetky udalosti zostali zachované, nie sme však už schopný určiť ich pôvodný rozsah, ani množstvo génov ktoré sa kedysi nachádzali na obrázku.

Kapitola 4

Porovnanie optimalizačných metód

V predchádzajúcej kapitole sme popísali, čo je to problém množinového pokrytia, a ako ho vieme využiť pre výber takej sady génov, ktorá nám zobrazí všetky udalosti, ktoré sa odohrali počas evolučnej histórie. Náš program obsahuje greedy algoritmus, ktorý nájde približné riešenie daného problému. Inou možnosťou je, previesť náš problém množinového pokrytia na celočíselný lineárny program, ktorého vyriešenia nám nájde optimálne riešenie. Pre greedy algoritmus poznáme jeho aproximačný koeficient $\ln m - \ln \ln m \ + \Theta(1)$ [5] kde $m = |U|$. Všetky próblemy množinového pokrytia, ktoré riešime majú spoločnú črtu, a to že vznikli ako prepis evolučnej histórie. Toto špecifikum môže mať vplyv na to, ako blízko k optimálnemu riešeniu sa priblíži greedy algoritmus. V tejto kapitole sa budeme venovať tomu, aké sú reálne rozdiely vo výsledkoch medzi aproximáciou a optimálnym riešením. Popíšeme spôsob akým testy prebiehajú, pozrieme sa na vyprodukované dáta, a porovnáme výsledky pre greedy algoritmus a celočíselné lineárne programovanie.

4.1 Priebeh testovania

Priebeh testovania vieme rozdeliť na tri časti. Prvou časťou je vytvorenie vstupných dát, na tento účel slúži pomocný program ktorý vytvorí požadované evolučné histórie. Druhým krokom je spracovanie týchto histórii, to znamená vyriešenie problému množinového pokrytia pre každú z nich pomocou greedy algoritmu ako aj pomocou celočíselného lineárneho programovania. Záverečná časť spočíva v zozbieraní získaných dát a ich porovnaní.

4.1.1 Generovanie evolučných histórií

Pre tento účel sme zostrojili jednoduchý program umožnujúci generovanie evolučných histórií. Tento program dostane na vstupe súbor s údajmi ako v tabuľke 4.1.2. Node_numb udáva koľko udalostí sa bude vo vygenerovanej histórii nachádzať a gene_numb

node_numb	10	
gene_numb	10	
len_rate	0.75	
duplication	1	
inversion	1	
deletion	1	3
insertion	1	5
translocation	1	

Tabuľka 4.1: Ukážka vstupu pre generátor histórií, settings.generator

určuje počet počiatočných génov. Každý krok evolučnej histórie bude obsahovať jednu udalosť, tú umiestnime na náhodné miesto do sekvencie génov, jej dĺžku, teda to koľko génov sa v tejto udalosti nachádza vypočítame na základe hodnoty len_rate. V udalosti s určitosťou ovplyvný jeden gén, každý ďalší pridáme s pravdebodobnosťou len_rate, jedná sa teda o geometrickú postupnosť. To, aká udalosť sa v danom kroku odohrá, určíme náhodne. Prvá hodnota za každou udalosťou z tabuľky 4.1.2, vyjadruje pomer šancí, že sa daná udalosť vyberie. V našom prípade sú udalosťi v pomere 1:1:1:1:1 mali by byť teda zastúpené rovnomerne. Pokiaľ niektorú z udalostí v histórii nechceme, nastavíme jej hodnotu 0. Pre inserciu a deléciu vieme navyše nastaviť maximálny počet génov ktoré ovplyvnia, keďže sú to udalosti ktoré menia počet génov v kroku (Duplikácia taktiež mení počet génov, ale vytvára iba kópie, nepridáva žiadne nové gény). Pri translokácii gény translokujeme na náhodné miesto. Okrem vstupného súboru, môžme generátoru zadať aj počet, koľko histórií má pre dané parametre vytvoriť.

Spracovanie vygenerovaných histórií Pre všetky vygenerované histórie následne spustíme náš hlavný program, vyexportujeme riešenie problému množinového pokrytia pomocou greedy algoritmu, ďalej vyexportujeme formuláciu celočíselného lineárne programu, a jeho riešenie nájdeme pomocou CPLEXu. Pre každú históriu si zapamätáme, aké množstvo blokov sme pokrývali a koľko génov sme na to mohli vuyžiť. Aký počet z týchto génov bol zvolený do greedy riešenia a aký do optimálneho riešenia CPLEXom. Posledným údajom budú časy, čas potrebný pre načítanie evolučnej histórie, čas ktorý potreboval greedy algoritmus pre nájdenie riešenia a čas pri hľadaní optima. Pre všetky histórie, ktoré generátor vytvoril z rovnakých nastavení, dáme tieto údaje dokopy. Získame priemerné hodnoty pre dané nastavenia generátora. To nám umožní sledovať vplyv nastavení generátora na výsledky testovania.

4.1.2 Ako zopakovať testovanie ?

Ak má čitateľ záujem zopakovať testovanie, tu je návod ako na to. K tejto práci, v priečinku *Testovanie* prikladám všetky súbory potrebné pre zopakovanie testovania. Konkrétne sú to tri java programy: bak.jar je hlavný program popísaný v tejto práci, spracováva histórie, hľadá greedy riešenie a exportuje zadanie celočíselného lineárneho programu. Generator.jar je popísaný v sekcii 4.1.1, na vstup dostáva súbor podobný tomu v tabuľke 4.1.2 a údaj koľko histórií z neho má vytvoriť. Pre zadaný vstup $filename.generator$ a počet n vytvorí histórie $filename\#1.history$,$filename\#2.history$ .. $filename\#n.history$. TestParser.jar je slúži k spracovaniu údajov pre všetky histórie ktoré vznikli z rovnakých nastavení generátora. Zoradí ich do CVS tabuľky - tabuľka v ktorej sú hodnoty oddelené medzeramy, každá história bude v jednom riadku, plus sa tu nachádza riadok *avg* s priemernými hodnotami. Automatizáciu testovania zabezpečuje Linuxový príkaz *make*. Vzťahy medzi súbormy, a spôsob akým *make* vytvára nové súbory je popísaný v súbore *Makefile*. Na začiatku musíme existovať súbor $filename.generator$. Prvé zavolanie príkazu *make* pre každý takýto súbor spustí generátor histórií. Údaj o tom, koľko histórií sa vygeneruje pre jedny nastavenia sa nachádza v *Makefile*. Druhé zavolanie príkazu *make* pre každú históriu $filename\#i.history$ vytvorenú v predchádzajúcom kroku, nájde greedy riešenia, optimálne riešenia za pomoci CPLEXu, a všetky výsledné údaje podstané pre testovanie uloží do súboru $filename\#i.grepped$. Tretie zavolanie príkazu *make* pre každý súbor $filename.generator$, z ktorého sme generovali histórie $filename\#1.history$,$filename\#2.history$ .. $filename\#n.history$, vytvorí *filename.data*. V tomto súbore sa nachádza tabuľka vytvorená programom *TestParser.jar*.

geny	udalosti	poradie	greedy	greedy-cas	ilp	ilp-cas
0	10	1	14	1.25	12	1.2
10	10	2	13	1.12	13	1.1
10	10	3	13	1.09	13	1.15
10	10	4	8	1.09	8	1.11
10	10	5	9	1.09	9	1.15
10	10	6	21	1.17	21	1.08
10	10	7	17	1.04	16	1.08
10	10	8	10	1.11	10	1.09
10	10	9	9	1.09	9	1.06
10	10	10	16	1.19	16	1.05
10	10	avg	13	1.124	12.7	1.107

Tabuľka 4.2: Ukážka vstupu pre generátor histórií, settings.generator

Záver

V závere je potrebné v stručnosti zhrnúť dosiahnuté výsledky vo vzťahu k stanoveným cieľom. Rozsah záveru je minimálne dve strany. Záver ako kapitola sa nečísluje.

Literatúra

[1] Albert Herencsár. An improved algorithm for ancestral gene order reconstruction. Master's thesis, Comenius University in Bratislava, 2014. Supervised by Broňa Brejová.

[2] Ján Hozza. Rekonštrukcia duplikačných histórií pomocou pravdepodobnostného modelu. Bachelor thesis, Comenius University in Bratislava, 2014. Supervised by Tomáš Vinař.

[3] R. Karp. Reducibility among combinatorial problems. In R. Miller and J. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[4] Jakub Kovac, Brona Brejova, and Tomas Vinar. A Practical Algorithm for Ancestral Rearrangement Reconstruction. In Teresa M. Przytycka and Marie-France Sagot, editors, *Algorithms in Bioinformatics, 11th International Workshop (WABI)*, volume 6833 of *Lecture Notes in Computer Science*, pages 163–174, Saarbrücken, Germany, September 2011. Springer.

[5] Petr Slavík. A tight analysis of the greedy algorithm for set cover. In *Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing*, STOC '96, pages 435–441, New York, NY, USA, 1996. ACM.

[6] Tomas Vinar and Brona Brejova. Biowiki, 2014.

[7] Tomas Vinar, Brona Brejova, Giltae Song, and Adam C. Siepel. Reconstructing Histories of Complex Gene Clusters on a Phylogeny. *Journal of Computational Biology*, 17(9):1267–1279, 2010. Early version appeared in RECOMB-CG 2009.

[8] Wikipedia. Linear programming — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Linear%20programming&oldid=713865251`, 2016. [Online; accessed 10-May-2016].

[9] David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, New York, NY, USA, 1st edition, 2011.

[10] M.J. Zvelebil and J.O. Baum. *Understanding Bioinformatics*. Garland Science, 2008.