

Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Richard Eliáš

Vizualizace sekundární struktury RNA s využitím existujících struktur

Katedra softwarového inženýrství

Vedoucí bakalářské práce: RNDr. David Hoksza, Ph.D.

Studijní program: Informatika

Studijní obor: Obecná informatika

Praha 2016

Prohlašuji, že jsem tuto bakalářskou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V dne

Podpis autora

Název práce: Vizualizace sekundární struktury RNA s využitím existujících struktur

Autor: Richard Eliáš

Katedra: Katedra softwarového inženýrství

Vedoucí bakalářské práce: RNDr. David Hoksza, Ph.D., Katedra softwarového inženýrství

Abstrakt: Abstrakt .. TODO

Klíčová slova: TODO klíčová slova

Title: RNA secondary structure visualization using existing structures

Author: Richard Eliáš

Department: Department of Software Engineering

Supervisor: RNDr. David Hoksza, Ph.D., Department of Software Engineering

Abstract: RNA secondary structure data, both experimental and predicted, are becoming increasingly available which is reflected in the increased demand for tools enabling their analysis. The common first step in the analysis of RNA molecules is visual inspection of their secondary structure. In order to correctly lay out an RNA structure, the notion of optimal layout is required. However, optimal layout of RNA structure has never been formalized and is largely habitual. To tackle this problem we propose an algorithm capable of visualizing an RNA structure using a related structure with a well-defined layout. The algorithm first converts both structures into a tree representation and then uses tree-edit distance algorithm to find out the minimum number of tree edit operations to convert one structure into the other. We couple each tree edit operation with a layout modification operation which is then used to gradually transform the known layout into the target one. The optimality of tree edit distance algorithm causes that the common motives are retained and the regions which differ in both the structures are taken care of. Visual inspection and planarity evaluation reveals that the algorithm is able to give good layouts even for relatively distant structures while keeping the layout planar. The new method is well suited for situations when one needs to visualize a structure for which a homologous structure with a good visualization is already available. ii

Keywords: RNA secondary structure, visualization, homology

Poděkování.

Obsah

Úvod	3
1 Úvod do štúdia RNA štruktúry a grafov	5
1.1 Čo je RNA	5
1.2 Sekundárna štruktúra	5
1.2.1 Typy zobrazenia sekundárnej štruktúry	6
1.2.2 Motívy	8
1.3 Hlavný objekt záujmu - rRNA	8
1.4 Stromová reprezentácia sekundárnej štruktúry	9
1.5 Grafové pojmy	12
2 Mapovanie medzi RNA stromami	14
2.1 Tree-edit-distance algoritmus	14
2.2 Vývoj algoritmu - rekurgia a dynamické programovanie	15
2.2.1 RTED	15
2.2.2 Mapovanie medzi stromami	20
3 Kreslenie molekuly	25
3.1 Štruktúry v RNA	25
3.2 Algoritmus	26
3.2.1 Normalizácia vzdialeností v báзовých pároch a vyrovnávanie stemov	26
3.2.2 Operácie na stromoch	26
3.2.3 Vkladanie nového vrcholu do stromu	27
3.2.4 Modifikácia multibrach loop	27
3.2.5 Mazanie vrcholu zo stromu	28
4 TRAVeLer - Template RnA Visualization	29
4.1 Inštalácia	29
4.2 Argumenty programu	29
4.2.1 Formát fasta súboru	30
4.3 Príklad vstupu	30
4.4 Výstupne súbory	32
4.5 Rozšírenie podpory iných vstupných obrázkov	32
5 Výsledky práce	35
5.1 Celkové výsledky	36
5.1.1 Otáčanie vetvy kvôli existujúcej hrane	38
5.1.2 Rozloženie báz na kružnicu	38
5.1.3 Otáčanie vetvy kvôli prekreslovaniu multibranch loopu	39
Záver	41
Seznam použité literatury	42
Zoznam obrázkov	44

Zoznam tabuliek	45
Seznam použitých zkratk	46
Přílohy	47

Úvod

Donedávna sa myslelo, že úloha ribonukleovej kyseliny, RNA, je obmedzená iba na syntézu bielkovín, buď ako nositeľka genetickej informácie (mRNA), alebo ako prenášač aminokyselín pri ich tvorbe (tRNA). Avšak existuje mnoho ďalších druhov, od relatívne malých molekúl majúcich iba desiatky nukleotidov, ktoré ovplyvňujú expresiu génov (miRNA, siRNA, snRNA, snoRNA a ďalšie) 3 10, až po veľké molekuly s tisíckami nukleotidov, ktoré sa podieľajú na tvorbe ribozómu (rRNA).

Spolu s objavmi ďalších funkcií RNA molekúl rastie záujem o nástroje dovoľujúce študovať štruktúru týchto molekúl. Primárna štruktúra je určená poradím nukleotidov v reťazci RNA. Priestorovým usporiadaním získame terciárnu štruktúru. Poslednou a v tejto práci pre nás najdôležitejšou bude sekundárna štruktúra. Tú reprezentuje zoznam nukleotidov ktoré sú spojené väzbou. Spárované nukleotidy sú blízko seba v priestore a tak nám sekundárna relatívne dobre aproximuje terciárnu štruktúru. Predpovedanie terciárnej štruktúry nieje veľmi spoľahlivé ani pre kratšie molekuly. Naopak, pre menšie molekuly a ich sekundárnu štruktúru existujú spoľahlivejšie metódy, ktorých prehľad a porovnanie nájdeme napríklad v 17. Príbuzné štruktúry nám vedú poslúžiť k predpovedaniu konzervovaných častí, dokonca aj veľkých rRNA molekúl 18.

Vizualizácia sekundárnej štruktúry RNA sa dá previesť na kreslenie grafu, ktorého vrcholy tvoria nukleotidy a hrany reprezentujú páry medzi nimi. Kreslenie grafov je značne preskúmanou témou, keďže nachádza uplatnenie vo veľa doménach, ako napríklad analýza sociálnych sietí 19 alebo vo všeobecnej analýze dát 21.

Cieľom vizualizácie sekundárnej štruktúry RNA je zachytiť párovanie nukleotidov v molekule a ideálne všetky ďalšie motívy, ktoré sa v štruktúre vyskytujú, ako napríklad hairpin, bulge, interior a multi-branch loop. Existujúce nástroje na vizualizáciu zväčša volia medzi tromi tipmi reprezentácie štruktúry 22: spojnicový graf (linked graph), kruhový graf (circular graph) alebo štandardná štruktúra (classical structure). Aj keď spojnicový a kruhový graf podporujú vizualizáciu párovania báz, motívy sa v nej dajú nájsť len veľmi ťažko, ak vôbec. Preto nám na hľadanie motívov v RNA ostala štandardná reprezentácia štruktúry RNA. Bolo vymyslených množstvo riešení - RNAfold z balíka ViennaRNA 14, VARNA 4, RnaViz 5, jViz.RNA 22, mfold 25, XRna ? , PseudoViewer 9 alebo RNAView 23. Avšak iba niektoré z týchto nástrojov a algoritmov sú použiteľné pre vizualizáciu veľkých štruktúr, akou sú napríklad podjednotky rRNA (RNAfold, RnaViz a RNAView). Porovnanie všetkých (až na mfold) nájdeme v článku 16.

Vzhľadom k tomu, že je nekonečne mnoho možností ako rozložiť sekundárnu štruktúru, potrebujeme zistiť aké kritéria by malo nakreslenie RNA splňovať. Nanešťastie tieto kritéria nie sú formalizované, avšak niektoré vlastnosti ako napríklad rovinnosť nakreslenia, kreslenie loopov (hairpin, bulge, interior a multi-branch) na kružnice a vrcholy stemu ležiace na jednej priamke sú spoločné pre väčšinu vizualizácií používaných vo vedeckej komunite 1. Ostatné sa prispôbujú oblasti štúdia, kvôli čomu každý algoritmus nebude vyhovovať veľkej časti používateľov. Dôvod môžeme ilustrovať na ribozomálnej RNA. Štruktúry týchto molekúl majú veľké konzervované časti, ktoré biológovia očakávajú na rovnakom

mieste na obrázku, vďaka čomu sa vedia orientovať aj v relatívne veľkej molekule a môžu skúmať a nachádzať tie menej konzervované časti, ktoré sa líšia medzi organizmami. To znamená, že ak chceme štruktúru vizualizovať, potrebujeme ukladať konzervované časti vždy na rovnaké miesto v obrázku.

Potreba vizualizácie, ktorá by zachovávala čo najviac spomínaných vlastností nás viedla k vytvoreniu nového vizualizačného algoritmu založeného na použití šablónovej (vzorovej) molekuly. Algoritmus na vstupe vezme cieľovú štruktúru, ktorú chceme vizualizovať a inú, podobnú štruktúru u ktorej poznáme jej nakreslenie. Túto podobnú molekulu nazývame šablónou. Obe štruktúry sú prevedené do ich stromovej reprezentácie. Následne nájdeme najkratšiu postupnosť editačných operácií, ktoré prevedú strom molekuly šablónovej na vizualizovanú molekulu a rovnako menia aj nakreslenie šablóny na nakreslenie cieľovej molekuly. Vzhľadom k tomu, že editačné operácie ktoré menia nakreslenie odpovedajú minimálnej editačnej vzdialenosti medzi stromami, nakreslenie spoločných častí sa nemení. Táto metóda je teda schopná vizualizovať sekundárnu štruktúru RNA molekuly presne podľa zvyku biológov - podľa poskytnutého vzorového nakreslenia.

1. Úvod do štúdia RNA štruktúry a grafov

V tejto časti práce zoznámime čitateľa s pojmami, ktoré s RNA a jej štruktúrou súvisia.

1.1 Čo je RNA

RNA, ribonukleová kyselina, je jednovláknová molekula, pozostávajúca zo sekvencie nukleotidov, jej základných stavebných častí. Tie sa ďalej skladajú z cukru (pentózy), dusíkatej bázy a zvyšku kyseliny fosforečnej. Poznáme štyri druhy báz, adenín, cytozín, guanín a uracyl, označovať ich budeme A, C, G, respektíve U. Okrem báz nás ďalšie zložky nebudú zaujímať a tak ďalej v texte stotožníme pojmy nukleotid a báza.

Štruktúru RNA môžeme chápať podľa stupňa zjednodušenia

- Primárna štruktúra - poradie nukleotidov v reťazci
- Sekundárna štruktúra - párovanie medzi bázami
- Terciárna štruktúra - priestorové usporiadanie molekuly

RNA ako jednovláknová molekula sa v snahe minimalizovať voľnú energiu páruje sama na seba. V tomto hrajú úlohu vodíkové väzby medzi nukleotidmi. Tie majú vzájomnú preferenciu, čo znamená, že páry vznikajú najčastejšie medzi A-U a C-G, no ani iné kombinácie nie sú vylúčené.

1.2 Sekundárna štruktúra

Hlavným objektom nášho záujmu je sekundárna štruktúra RNA. V nasledujúcej časti si ju definujeme formálnejšie.

Definícia 1. *Primárna štruktúra RNA je určená poradím nukleotidov v polynukleotidovom reťazci.*

Sekundárnou štruktúrou označíme množinu \mathbb{S} párov báz (i, j) takých, že pre dva páry (i, j) a $(k, l) \in \mathbb{S}$ (bez straty všeobecnosti $i \leq k$) platí jedno z nasledujúcich:

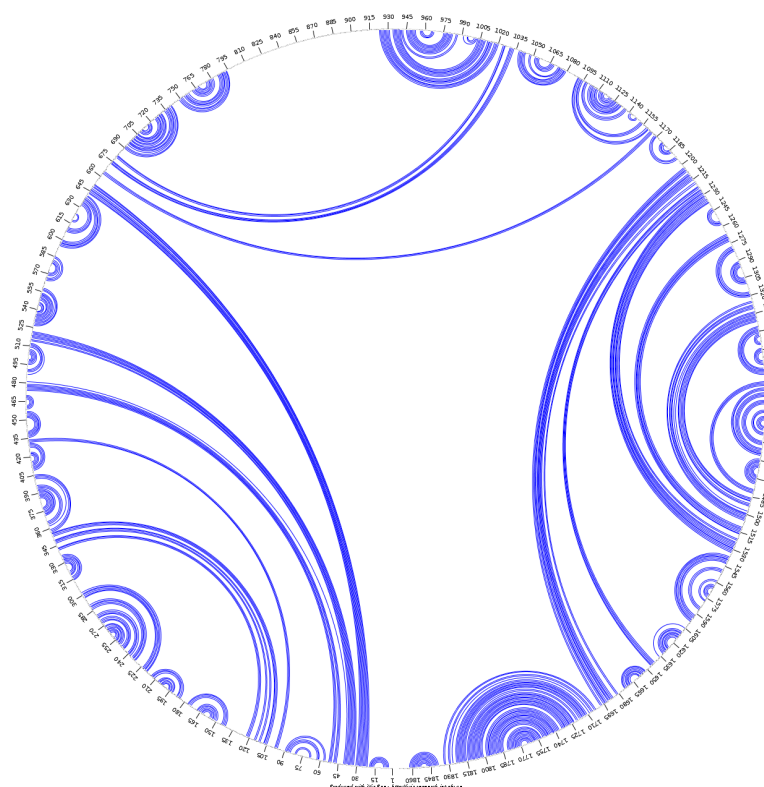
- $i = k \iff j = l$
- $i < j < k < l$, čiže pár (i, j) predchádza pár (k, l)
- $i < k < l < j$, čiže pár (i, j) obsahuje pár (k, l)

Prvá podmienka zabezpečuje, že nukleotid je najviac v jednom bázičkom páre, druhá a tretia hovoria o ich usporiadaní - buď na seba nadväzujú, alebo sú na sebe nezávislé.

Všimnime si ďalej, že podmienky vylučujú bázoové páry typu (i, j) a (k, l) , kde $i < k < j < l$, teda páry sa nesmú prekrývať. Takéto páry nazývame pseudouzly (pseudoknot) a ich rozdelenie máme na obrázku 1.5. Pseudouzly sa často



Obr. 1.1: Spojnicový graf



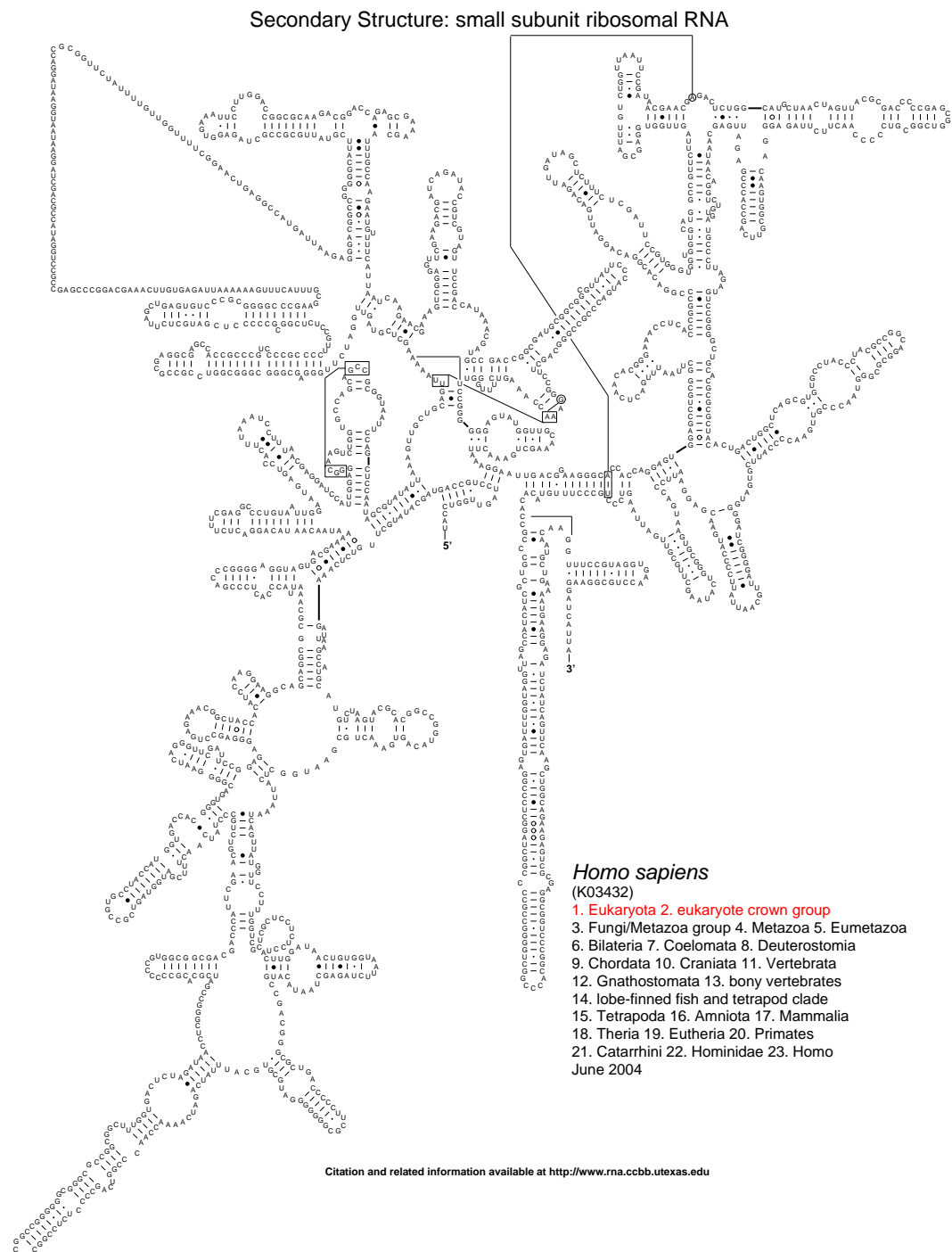
Obr. 1.2: Kruhový graf

považujú za súčasť sekundárnej štruktúry, no v našej práci s nimi nepočítame. Takéto zjednodušenie nám umožní reprezentovať sekundárnu štruktúru RNA ako usporiadaný zakorenený strom (les). Definíciu grafových pojmov ako strom a les uvedieme neskôr.

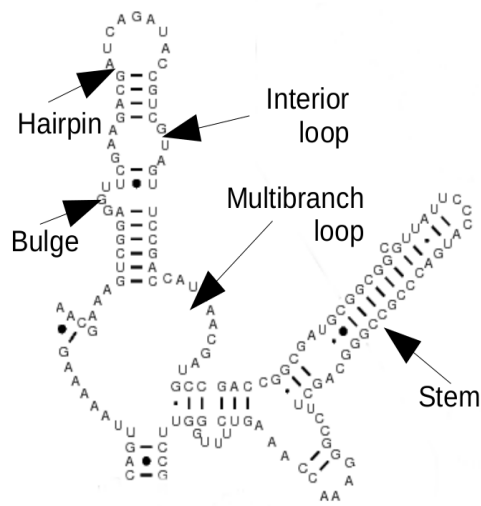
1.2.1 Typy zobrazenia sekundárnej štruktúry

Existujúce nástroje volia medzi tromi možnosťami vizualizácie sekundárnej štruktúry: spojnicový graf (linked graph), kruhový graf (circular graph) alebo štandardná štruktúra (classical structure).

Na nasledujúcich obrázkoch sme pomocou programu jViz.Rna 22 nakreslili molekulu K03432 malej podjednotky ribozomálnej RNA človeka vo všetkých zobrazeniach. Ako vidíme, zo spojnicového 1.1 a rovnako aj kruhového grafu 1.2 sa motívy vyčítať tak ľahko nedajú. Naopak zobrazenie štandardnej štruktúry na obrázku 1.3 podľa predstáv biológov. Tí očakávajú, že v RNA molekulách rovnakého typu (napríklad malé podjednotky rRNA 23S) budú konzervované časti nakreslené na rovnakom mieste, čo im pomôže orientovať sa aj pri molekulách veľkých rozmerov.



Obr. 1.3: Vizualizácia sekundárnej štruktúry RNA človeka z CRW databázy 2



Obr. 1.4: Štruktúrne motívy v RNA



Obr. 1.5: Typy pseudouzlov podľa 12

1.2.2 Motívy

Niektoré časti molekuly vytvárajú isté motívy, z ktorých niektoré môžeme vidieť na obrázku 1.4. Motívom sú aj pseudouzly 1.5, ale v našej práci ich úplne ignorujeme.

Stem (stonka) je časť molekuly, kde sa na seba párujú dva súvislé časti vlákna. Loopom budeme označovať miesta s nespárovanými bázami. Bulge (vypuklina) je miesto v steme, ktoré z jednej strany obsahuje loop. Podobná je interior (vnútorná) loop, tá ale má loop po oboch stranách stemu. Hairpin je stem zakončený loopom a nakoniec multibranch (viac vetvová) loop je podobná ako interior loop, ale spája dokopy viac stemov. V ďalšom rozprávaní nám bude stačiť rozdelenie na stem (spárované bázy) a loop (nespárované bázy).

1.3 Hlavný objekt záujmu - rRNA

Ako hlavný objekt záujmu sme si spomedzi všetkých druhov RNA vybrali práve ribozomálnu, rRNA. Jej funkcia je hlavne v translácii génov, pri ktorej sa genetická informácia prekladá z poradia nukleotidov v mRNA do poradia aminokyselín v bielkovine.



Obr. 1.6: Malá podjednotka vygenerovaná programom jViz.Rna 22

Ribozomálnu RNA sme zvolili kvôli jej konzervovanosti naprieč evolučným spektrom. Ďalším dôvodom bola jej veľkosť, ktorá robí existujúcim nástrojom najväčšie problémy pri vizualizácii. Na obrázku 1.3 vidíme sekundárnu štruktúru K03432 malej podjednotky ribozomálnej RNA človeka z CRW databázy. Tá znázorňuje predstavy biológov o tom, ako by malo nakreslenie danej molekuly vyzerieť. Obrázky 1.6, 1.7 a 1.8 sú zase vygenerované vizualizácie pomocou programov jViz.Rna, Mfold a RNAfold. Dodržiavanie základných kritérií ako rovinnosť, loopy na kružniciach a stemy na priamkach sa viac, či menej programom darí, no celkový vzhľad obrázkov je úplne odlišný od požiadavok biológov, keďže sa motívy z obrázka z CRW databázy hľadajú veľmi ťažko.

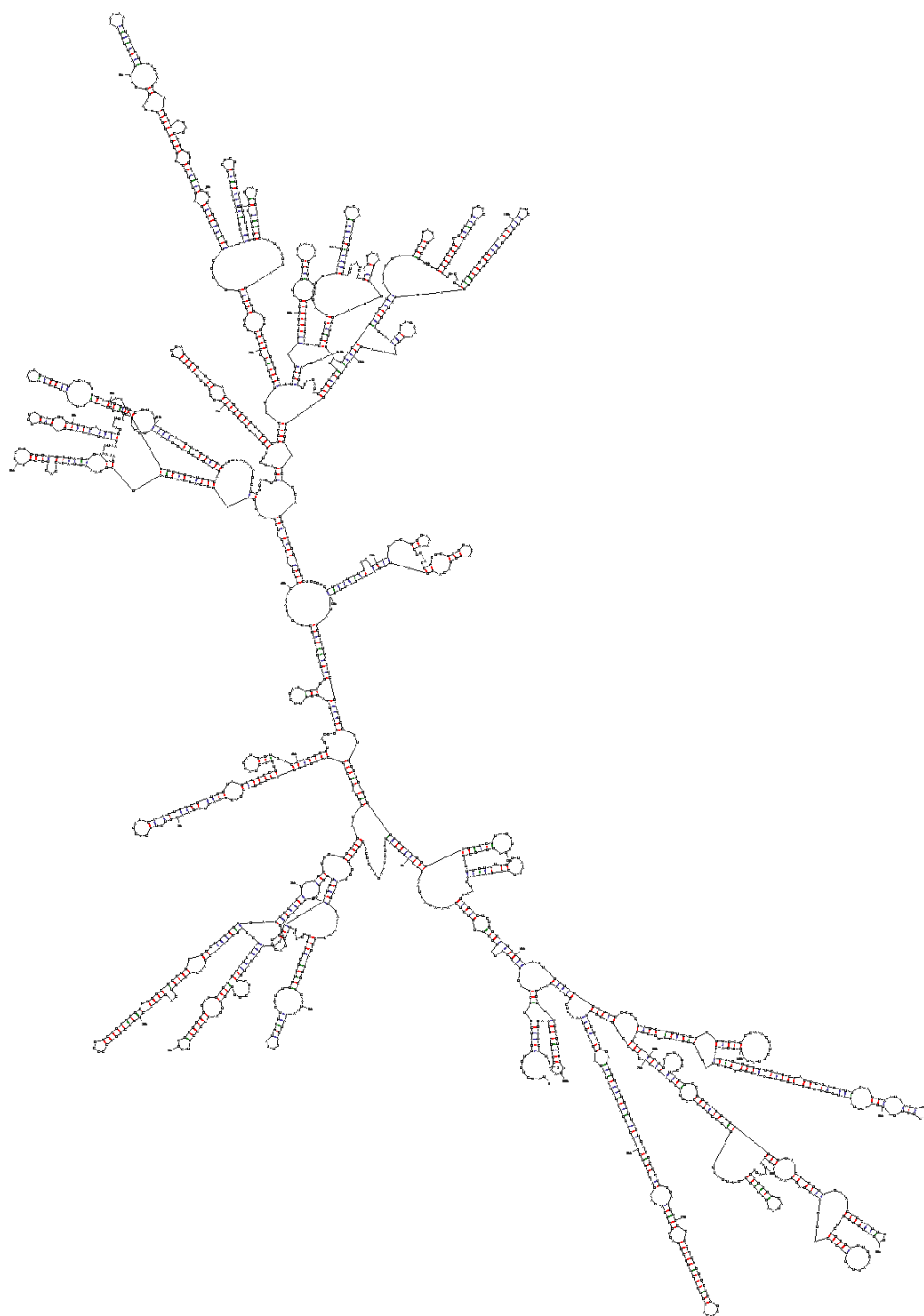
1.4 Stromová reprezentácia sekundárnej štruktúry

Vďaka ignorovaniu pseudouzlov v definícii 1 sekundárnej štruktúry ju môžeme reprezentovať ako usporiadaný strom ¹.

Bez straty všeobecnosti budeme vždy hovoriť ako o strome, aj keď sa môže stať (a typicky aj stáva), že štruktúra nebude celistvá, teda nejde o strom, ale o

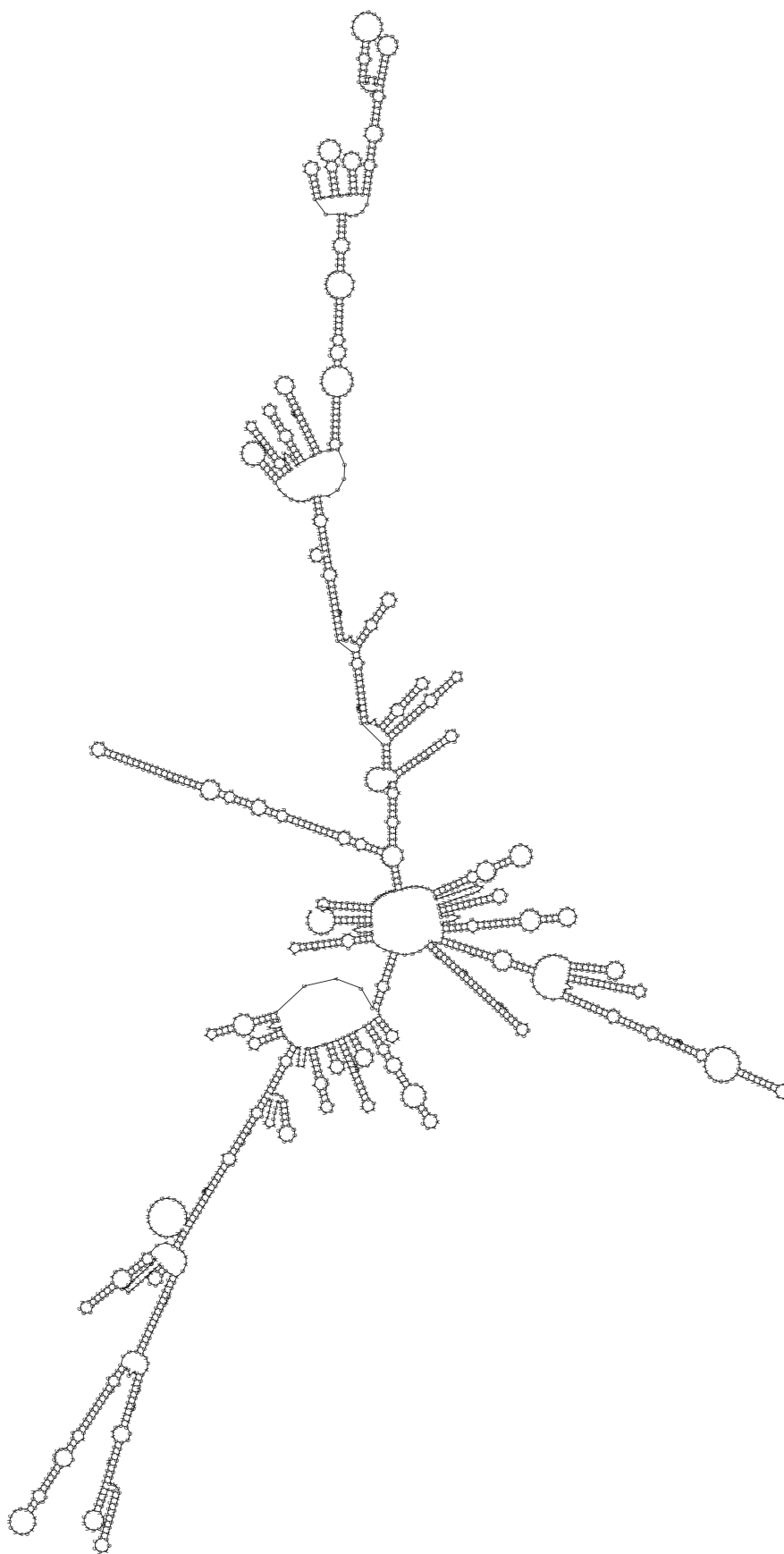
¹Používame pojmy z nasledujúcej kapitoly Grafové pojmy

Output of sir_graph (©)
mob_util 4.7

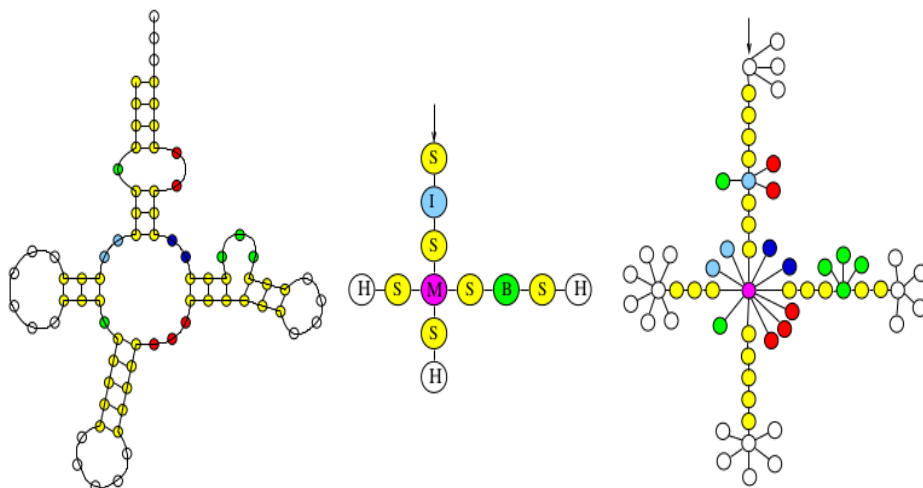


$dG = -692.73$ [Initially -763.70] human

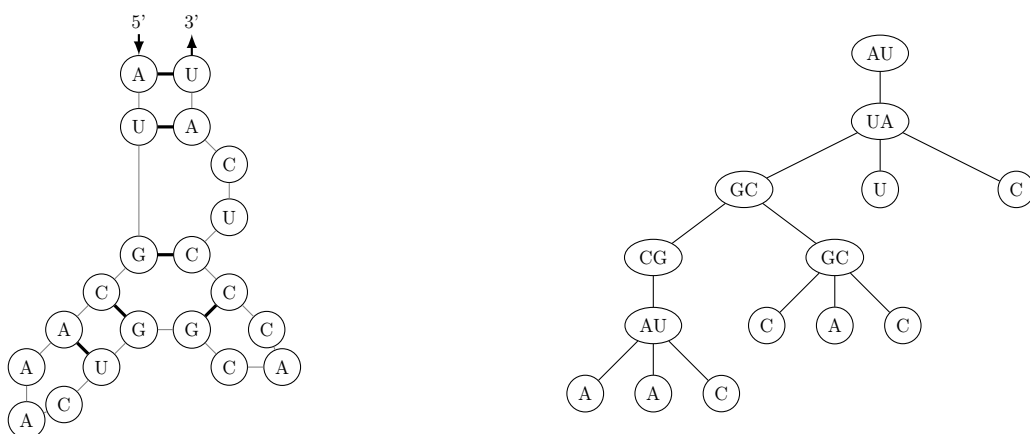
Obr. 1.7: Malá podjednotka vygenerovaná programom Mfold 25



Obr. 1.8: Malá podjednotka vygenerovaná programom RNAfold 14



Obr. 1.9: Varianty reprezentácie vrcholov podľa 1



Obr. 1.10: Molekula RNA a jej stromová reprezentácia

les. V tom prípade ale môžeme pripojiť nový špeciálny koreňový vrchol, ktorého potomkovia (viz. definícia 3) budú dané stromy a ktorý nebudeme vizualizovať.

Na obrázku 1.9 vidíme varianty reprezentácie vrcholov. Tie môžu zastupovať celé motívy, alebo iba nukleotid, respektíve celý básový pár.

V našej práci vrchol stromu reprezentuje básový pár (vnútorný vrchol) alebo nespárovanú bázu (list stromu), ako je to zobrazené na obrázku 1.10. Štruktúru do ktorej patrí si totiž vieme ľahko zistiť z potomkov vrcholu.

1.5 Grafové pojmy

V tejto časti zdefinujeme pojmy a značenie, ktoré budeme používať naprieč celou prácou. Z väčšej časti ho prevezmeme od Pawlik a Augsten (2011).

Definícia 2. *Usporiadáný zakorenený strom je orientovaný graf bez cyklov, jeho hrany sú orientované vždy v smere od koreňa, t.j. z predka k potomkovi.*

Okrem koreňa má každý vrchol svojho predka a existuje usporiadanie medzi potomkami.

Usporiadany les je usporiadaná množina stromov.

Ak F je les, V_F budeme označovať množinu jeho vrcholov a E_F množinu jeho hrán. Prázdny strom/les budeme značiť \emptyset .

Definícia 3. *Nech F je strom, u a v jeho dva rôzne vrcholy. Hovoríme, že u je predkom (rodičom) v (v je potomok u) ak $(u, v) \in E_F$. Hovoríme, že u je súrodencom v , ak sú to rôzne vrcholy a majú spoločného predka.*

Vnútrotnými vrcholmi budeme označovať vrcholy, ktoré majú nejakého potomka, tie bez potomkov nazveme listy.

Podles lesa F je les G s vrcholmi $V_G \subseteq V_F$ a hranami $E_G \subseteq E_F \cap (V_G \times V_G)$. Obdobne to platí aj pre podstrom stromu.

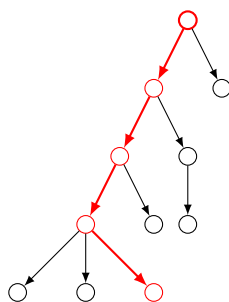
Nech v je vrchol stromu F . Potom F_v budeme značiť podstrom F zakorenený vo v , t.j. v strome ostávajú potomkovia v , ich potomkovia, atď. $F - v$ budeme značiť les, ktorý vznikne zmazaním vrcholu v z F spolu so všetkými hranami obsahujúcimi v . Podobne $F - F_v$ budeme značiť les, ktorý dostaneme zmazaním podstromu F_v z F .

Definícia 4 (Root-leaf cesta). *Nech je F strom. Cestou v F nazveme jeho súvislý podgraf taký, že všetky jeho vrcholy majú stupeň najviac 2.*

Root-leaf cestou v F budeme nazývať cestu začínajúcu v koreni stromu a končiacu v nejakom jeho liste.

V práci budeme využívať tri druhy ciest, budú to *left*, *right*, *heavy* a označovať ich budeme $\gamma^L, \gamma^R, \gamma^H$. *Left* bude v strome cesta idúca vždy do prvého (ľavého) potomka, *right* do posledného (pravého) a *heavy* zase cesta idúca do vrcholu s najväčším podstromom (najťažším)².

Príklad *root-leaf* cesty v strome je na obrázku 1.11.



Obr. 1.11: Príklad stromu. Je v ňom červene vyznačená *heavy* cesta

²ťažkých ciest môže existovať aj viac ako jedna

2. Mapovanie medzi RNA stromami

Ako sme spomínali v predchádzajúcich častiach, biológovia očakávajú, že podobné RNA molekuly (pozeráme na sekundárnu štruktúru) budú mať aj podobnú vizualizáciu. To znamená, že nakreslenia sa majú líšiť iba v rozdielnych častiach.

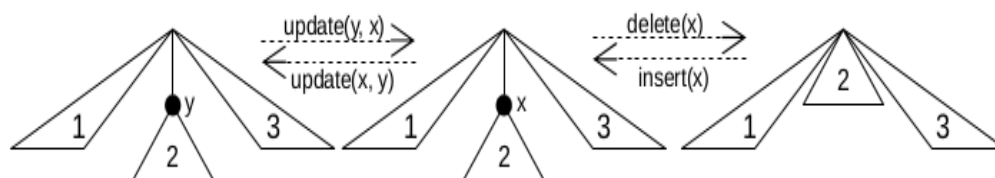
Vieme už, že RNA a jej sekundárnu štruktúru vieme reprezentovať ako usporiadaný strom. Táto kapitola nám dá návod, ako nájsť najmenší počet úprav, ktorý prevedie jeden strom na iný. Vďaka tomu, že poznáme vizualizáciu jednej molekuly - vieme rozvrhnutie jej báz na obrázku a dokážeme ju previesť na cieľovú molekulu, rovnaké štruktúry budú rovnako vizualizované.

To nás privádza k algoritmu tree-edit-distance, TED. Je obdobou Levenshteinového string-edit-distance algoritmu 13. Ten počíta editačnú vzdialenosť medzi dvomi reťazcami a transformuje jeden reťazec na druhý.

Problém u reťazcov je špeciálnym prípadom TEDu, kedy nám stromy zdegenerovali na cesty (spojový zoznam).

2.1 Tree-edit-distance algoritmus

Základ TED algoritmu je v rekurzívnom vzorci 2.2. Vzdialenosť medzi stromami F a G , $\delta(F, G)$ je definovaná ako minimálny počet editačných operácií, ktoré z F urobia G . Používame štandardne editačné operácie - delete, insert, update.



Obr. 2.1: Ukážky TED operácií

Delete, operácia zmazania vrcholu, znamená pripojiť k jeho predkovi všetkých jeho potomkov so zachovaním poradia medzi nimi. Insert, vkladanie vrcholu, je opačná operácia. Vkladáme vrchol medzi predka a nejakých jeho, po sebe nasledujúcich potomkov. Update iba zmení hodnotu vo vrchole stromu. Tieto operácie máme ukázané na obrázku 2.1.

Definícia 5 (Editačná vzdialenosť). *Nech F a G sú dva stromy. Editačná vzdialenosť (tree-edit-distance, $\delta(F, G)$), medzi F a G je rovná minimálnej cene, za ktorú strom F transformujeme na G .*

Rekurzia 2.2 počíta vzdialenosť medzi dvoma lesmi F a G . c_{del} , c_{ins} a c_{upd} sú ceny zmazania, vloženia a updatu vrcholu v strome a r_F a r_G sú korene F , respektíve G a to buď obidva najpravejšie alebo najľavejšie (tzn. vyberieme najpravejší/najľavejší strom lesa a jeho koreň).

$$\begin{aligned}
\delta(\emptyset, \emptyset) &= 0 \\
\delta(F, \emptyset) &= \delta(F - r_F, \emptyset) + c_{del}(r_F) \\
\delta(\emptyset, G) &= \delta(\emptyset, G - r_G) + c_{ins}(r_G) \\
\delta(F, G) &= \begin{cases} \delta(F - r_F, G) + c_{del}(r_F) \\ \delta(F, G - r_G) + c_{ins}(r_G) \\ \delta(F - F_{r_F}, G - G_{r_G}) \\ \quad + \delta(F_{r_F} - r_F, G_{r_G} - r_G) \\ \quad + c_{upd}(r_F, r_G) \end{cases}
\end{aligned}$$

Obr. 2.2: Rekurzívny vzorec pre výpočet tree-edit-distance od Demaine a kol. (2009) a Pawlik a Augsten (2011)

2.2 Vývoj algoritmu - rekurzia a dynamické programovanie

Algoritmus TED prešiel vývojom postupne od Tai (1979), ktorý predstavil algoritmus na výpočet editačnej vzdialenosti s priestorovou a časovou zložitou $\mathcal{O}(m^3 \cdot n^3)$. Algoritmus vylepšili Zhang a Shasha (1989) vypočítaním toho, že nepotrebuje počítať celú rekúziu. Ich algoritmus má časovú zložitou $\mathcal{O}(m^2 \cdot n^2)$ a priestorovú $\mathcal{O}(m \cdot n)$. Klein (1998) dosiahol časovú zložitou $\mathcal{O}(m^2 \cdot n \cdot \log n)$, avšak jeho riešenie potrebuje rovnako veľa pamäte. Dulucq a Touzet (2003) ukázali, že minimálny čas na beh algoritmu je $\mathcal{O}(m \cdot n \cdot \log m \cdot \log n)$. Demaine a kol. (2009) predstavili worst-case optimálny algoritmus pre tree-edit-distance. Jeho časová a priestorová zložitou je $\mathcal{O}(m^2 \cdot n \cdot (1 + \log \frac{n}{m}))$ a $\mathcal{O}(m \cdot n)$. Pawlik a Augsten (2011) ukázali spojitou medzi efektívnosťou predchádzajúcich algoritmov a tvarom stromov. Zovšeobecnil predchádzajúce prístupy a vytvorili algoritmus s worst-case časom $\mathcal{O}(m^3)$ a priestorom $\mathcal{O}(m \cdot n)$. Ako dokázali, ich algoritmus je efektívny pre všetky tvary stromov a worst-case prípad u neho nastane iba vtedy, ak lepší smer výpočtu neexistuje.

2.2.1 RTED

V nasledujúcich častiach sa budeme venovať výhradne algoritmu RTED od tvorcov Pawlik a Augsten (2011). Ich algoritmus rozdelíme na 2 časti, rovnako pomenovaný RTED a GTED.

RTED (Robust Tree Edit Distance) algoritmus bude pre nás algoritmus na výpočet optimálnej dekompozičnej stratégie (viz definícia 6) a GTED (General Tree Edit Distance), algoritmus pre samotný výpočet editačnej vzdialenosti z rekúzie 2.2 s aplikovaním danej stratégie.

Definícia 6 (Dekompozičná stratégia). *Nech F a G sú lesy. Dekompozičná stratégia pre rekúziu 2.2 priradí každej dvojici stromov F_v a G_w jednu cestu γ_T z koreňa do listu, kde $T \in \{F, G\}$.*

LRH dekompozičná stratégia vyberá vždy najľavejší/najpravejší/najťažší (left/right/heavy) vrchol až kým nepríde do listu. Najťažší vrchol je taký, v ktorého podstrome je najviac vrcholov.

To znamená, že dekompozičná stratégia nám pre dvojicu lesov povie, ktorý z nich a akou cestou chceme rozložiť (dekomponovať). V rekurzii to hovorí, či odoberám r_F , alebo r_G vrchol.

GTED: General Tree Edit Distance algoritmus

Začneme princípom fungovania GTED algoritmu. Detaily pre LRH stratégie sú v Zhang a Shasha (1989) pre left/right a v Demaine a kol. (2009) pre heavy stratégiu.

Definícia 7. *Relevant subtrees stromu F pre root-leaf cestu γ sú stromy $F - \gamma$. Relevant subforests lesa F pre nejakú root-leaf cestu γ sú definované rekurzívne (r_R a r_L označujú najpravejší, respektíve najľavejší koreň F)*

$$\begin{aligned}\mathcal{F}(\emptyset, \gamma) &= \emptyset \\ \mathcal{F}(F, \gamma) &= \{F\} \cup \begin{cases} \mathcal{F}(F - r_R(F), \gamma), & \text{ak } r_L(F) \in \gamma \\ \mathcal{F}(F - r_L(F), \gamma), & \text{v ostatných prípadoch} \end{cases}\end{aligned}$$

Algorithm 1 General Tree Edit Distance for LRH strategies

```

1: procedure GTED( $F, G, TreeDistance, Strategies$ )
2:    $\gamma \leftarrow \text{path in } Strategies[F, G]$ 
3:   if  $\gamma \in F$  then
4:     for all tree  $F' \in F - \gamma$  do
5:        $TreeDistance \leftarrow TreeDistance$ 
6:        $\cup \text{GTED}(F', G, TreeDistance, Strategies)$ 
7:      $TreeDistance \leftarrow TreeDistance$ 
8:      $\cup \text{COMPUTEDISTANCE}(F, G, TreeDistance, \gamma)$ 
9:   else
10:     $TreeDistance \leftarrow TreeDistance$ 
11:     $\cup (\text{GTED}(G, F, TreeDistance^T, Strategies^T))^T$ 
12:   return  $TreeDistance$ 

```

Poznámka. Funkcia *OrderedSubforests* v algoritme 2 vracia zoradené podlesy daného lesa v opačnom poradí, ako ich pridávame v definícii 7.

GTED algoritmus 1 funguje v troch krokoch. Najprv podľa stratégie a ňou určenej cesty γ dekomponuje jeden zo stromov, môžeme si predstaviť, že je to práve F . Následne rekurzívne spočíta editačnú vzdialenosť medzi všetkými stromami, ktoré susedia s dekompozičnou cestou (t.j. $F - \gamma$) a stromom G .

Následne pre všetky relevant-subtrees stromy G' stromu G vyráta vzdialenosti medzi F_v a G' , ktorá dopočíta vzdialeností medzi vrcholmi $v \in \gamma_F$ a stromami G' .

Algorithm 2 Single path function

```
1: procedure COMPUTEDISTANCE( $F, G, TreeDistance, \gamma$ )
2:   if  $\gamma \in \gamma^*(F)$  then
3:     for all  $G' \in \text{RELEVANTSUBTREES}(G, \gamma)$  do
4:        $\text{SINGLEPATH}(F, G', TreeDistance, \gamma)$ 
5:   else
6:     for all  $F' \in \text{RELEVANTSUBTREES}(F, \gamma)$  do
7:        $\text{SINGLEPATH}(F', G, TreeDistance, \gamma)$ 

8: procedure SINGLEPATH( $F, G, TreeDistance, \gamma$ )
9:    $ForestDistance \leftarrow \text{array } (|F| + 1) \times (|G| + 1)$ 
10:   $ForestDistance[\emptyset][\emptyset] := 0$ 
11:  for  $F'$  subforest in  $\text{ORDEREDSUBFORESTS}(F, \gamma)$  do
12:     $Last_F \leftarrow \text{last added node to } F'$ 
13:     $ForestDistance[F'][\emptyset] := ForestDistance[F' - Last_F][\emptyset]$ 
14:     $+ C_{del}(Last_F)$ 
15:  for  $G'$  subforest in  $\text{ORDEREDSUBFORESTS}(G, \gamma)$  do
16:     $Last_G \leftarrow \text{last added node to } G'$ 
17:     $ForestDistance[\emptyset][G'] := ForestDistance[\emptyset][G' - Last_G]$ 
18:     $+ C_{ins}(Last_G)$ 
19:  for  $F'$  subforest in  $\text{ORDEREDSUBFORESTS}(F, \gamma)$  do
20:    for  $G'$  subforest in  $\text{ORDEREDSUBFORESTS}(G, \gamma)$  do
21:       $Last_F \leftarrow \text{last added node to } F'$ 
22:       $Last_G \leftarrow \text{last added node to } G'$ 
23:      if both  $F'$  and  $G'$  are trees then
24:         $C_{min} := \min \{$ 
25:           $ForestDistance[F' - Last_F][G'] +$ 
26:           $C_{del}(Last_F),$ 
27:           $ForestDistance[F'][G' - Last_G] +$ 
28:           $C_{ins}(Last_G),$ 
29:           $ForestDistance[F' - Last_F][G' - Last_G] +$ 
30:           $C_{upd}(Last_F, Last_G)$ 
31:           $ForestDistance[F', G'] := C_{min}$ 
32:           $TreeDistance[Last_F][Last_G] := C_{min}$ 
33:        else
34:           $C_{min} := \min \{$ 
35:             $ForestDistance[F' - Last_F][G'] +$ 
36:             $C_{del}(Last_F),$ 
37:             $ForestDistance[F'][G' - Last_G] +$ 
38:             $C_{ins}(Last_G),$ 
39:             $ForestDistance[F' - F_{Last_F}][G' - G_{Last_G}] +$ 
40:             $TreeDistance[F_{Last_F}][G_{Last_G}]\}$ 
41:           $ForestDistance[F'][G'] := C_{min}$ 
42:  return  $ForestDistance$ 
```

Lemma 1. *Ak `ComputeDistance` funkcia dopočíta editačnú vzdialenosť medzi vrcholmi na ceste γ a všetkými podstromami druhého stromu, potom `GTED` vráti maticu vzdialenosti medzi všetkými dvojicami podstromov F_v a G_w , pre $v \in F; w \in G$.*

Dôkaz. Nech $\gamma \in F$. Po vyrátaní editačnej vzdialenosti medzi stromami $F - \gamma$ a G nám stačí dopočítať už len vrcholy na ceste, teda vzdialenosti medzi stromami F_v a G pre $v \in \gamma_F$. □

Vďaka dôslednému usporiadaniu lesov si v každom kroku pripravíme potrebné dáta pre ďalší krok algoritmu 2.

Pozrime sa znovu na algoritmus a na hodnoty používané v podmienkach na riadkoch 23 a 33. Prvé dva sú v oboch rovnaké. Počítame hodnotu zmazania a vloženia vrcholu z /do F . Tretia hodnota sa líši podľa toho, či sú lesy zároveň stromami. Ak sú, tak na danom mieste je cena namapovania podstromov $F_v - v$ na $G_w - w$ a updatu vrcholu v na w . Ináč, keď aspoň jeden z lesov nieje strom, tak cenu mapovania medzi F_{Last_F} a G_{Last_G} máme vyrátanú z predchádzajúcich krokov, alebo z inej vetvy rekurzie. Následne nastavíme hodnotu vzdialenosti medzi lesmi na minimum a v prípade že sú to obidva stromy, tak si uložíme aj ich vzdialenosť.

Poznámka. Nikdy nepoužívam viackrát rovnakú cestu γ v strome. To vyplýva z toho, že po dekompozícií stromu podľa γ , iné stromy cestu γ neobsahujú.

Poznámka. `Single-path` funkcia každú hodnotu `ForestDistance`, rovnako ako `TreeDistance` nastavuje práve raz.

Dôkaz. Žiadnu cestu nepoužívam opakovane. Hodnotu v `TreeDistance` nastavujem iba v momente, keď sú obidva lesy stromami (teda ich korene ležia na cestách γ_F a γ_G) a to sa udeje práve raz. Lesy vždy iba zväčšujem, takže nikdy sa nedostanem do menšieho aby som mu mohol znovu nastaviť hodnotu. To isté platí aj pre `ForestDistance`. □

Lemma 2. *Nikdy nepoužívame neinicializované hodnoty `TreeDistance` a `ForestDistance`.*

Dôkaz. Hodnota `ForestDistance` pre použitie s prázdny lesom je inicializovaná, a pri každej iterácii algoritmu čítam iba z hodnôt z predchádzajúcich iterácií, napr. `ForestDistance` $[F - Last_F][G - Last_G]$, alebo `ForestDistance` $[F - F_{Last_F}][G - G_{Last_G}]$. V prvom prípade mažem iba jeden vrchol, v druhom celý jeho podstrom.

Hodnoty `TreeDistance` používame iba v prípade, že aspoň jeden z lesov F' alebo G' nieje stromom. To znamená, že ak posledne pridaný vrchol $Last_F$ je mimo cesty γ_F , tak sme vzdialenosť od $Last_G$ vyrátali rekurzívne po dekompozícií F už skôr. Naopak ak $Last_F$ leží na ceste, potom $Last_G$ je mimo cesty, a editačnú vzdialenosť sme vyrátali pri počítaní relevant-subtrees. □

F:

G:

<i>ForestDistance</i> :			
	0	1	2
	1	2	1
	2	1	2
	3	2	1

Obr. 2.3: Príklad výpočtu GTED medzi stromami F a G

Dôsledok. Algoritmus funguje.

Dôkaz. V predchádzajúcich častiach sme dokázali, že v každom kroku používame iba korektné hodnoty a všetky časti algoritmu počítajú správne, takže algoritmus GTED je v poriadku. □









Na záver tejto časti ukážeme na príklade, ako náš program TRAVeLer počíta vzdialenosť medzi dvoma stromami. Budeme používať, rovnako ako aj vo vyvinutej aplikácii, konštanty $c_{del} = c_{ins} = 1$, $c_{upd} = 0$. Sú nastavené tak kvôli tomu, že sa snažíme minimalizovať počet štrukturálnych zmien v strome, teda počet vkladaní a mazaní a zmena bázy nám vizualizáciu nemení. Obrázok 2.3 nám znázorňuje tabuľku *ForestDistance* z algoritmu 1 po skončení výpočtu.

Pozrime sa bližšie, ako algoritmus postupuje s výpočtom. Predpokladajme, že používame left stratégiu, ktorá nám určí cestu $\gamma = \{A, C\} \in G$. Hneď na začiatku algoritmus rozloží strom G podľa tejto cesty a rekurzívne spracuje $\{B\}^1 \in G$ (jeho podstrom) a vyráta vzdialenosť medzi ním a stromom F .

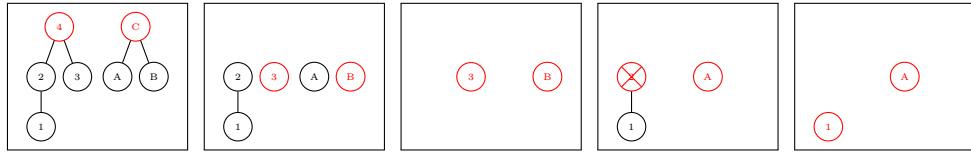
Následne začne vykonávať *SinglePath* funkciu 2, ktorej úloha je iba dorátať vzdialenosti medzi vrcholmi na ceste γ a druhým stromom. Výpočet je uvedený v tabuľke na obrázku 2.3. Začne porovnávať dva lesy, $\{1\}$ a $\{A\}$. Najmenšiu vzdialenosť (0) získame operáciou $\text{update}(1, A)$, teda zmeníme hodnotu vo vrchole z 1 na A. Vzdialenosť si uložíme do *TreeDistance* tabuľky, keďže podmienka, aby boli obidva lesy aj stromami platí. Pokračujeme porovnávaním lesov $\{1\}$ a $\{A, B\}$. Máme tri možnosti:

- zmažeme vrchol 1 a namapujeme $\{\}$ na G
- zmažeme vrchol B a namapujeme $\{1\}$ na $\{A\}$
- použijeme mapovanie, ktoré už máme v tabuľke $\{1\}$ na $\{B\}$ (z *TreeDistance*) a namapujeme $\{\}$ na $\{A\}$ (z *ForestDistance*)

¹Množinou vrcholou budeme priamo označovať les/strom indukovaný týmito vrcholmi

<i>TreeDistance</i> :			
	A	B	
	0	0	2
 	1	1	1
	0	0	2
  	3	3	1

Obr. 2.4: Výsledná tabuľka *TreeDistance* medzi stromami z 2.3



Obr. 2.5: Výpočet mapovania medzi stromami

Hodnotu z tretej podmienky máme uloženú z predchádzajúcich krokov (tento prípad sme vyrátali pri rekurzívnom počítaní vzdialenosti medzi $\{B\}$ a celého stromu F). Ďalej pokračujeme podobne, až kým dorátame všetky hodnoty. Výsledná tabuľka je na obrázku 2.4 a určuje ceny transformácie medzi stromami.

2.2.2 Mapovanie medzi stromami

Keď už máme vyrátanú *TreeDistance* tabuľku, môžeme sa pustiť do mapovania medzi stromami. Tabuľku používame iba vo funkcii *SinglePath* a na samotné mapovanie používame *ForestDistance*, ktorá nám hovorí veľa o štruktúre stromov/lesov s ktorými pracujeme.

Princípom počítania mapovania je spätné prechádzanie matice *ForestDistance*, teda zisťujeme, akú operáciu sme v ktorom bode zvolili (update, delete, insert).

Všimnime si, že algoritmus 3 prechádza postupne relevantné podlesy (relevant subforests z definície 7) vďaka výberu vrcholov v a w a tie mapuje na seba.

Na obrázku 2.5 pokračujeme v príklade a chystáme sa namapovať tieto dva stromy na seba. Červenou sú kreslené vrcholy v a w z algoritmu. Tretí obrázok predstavuje rekurzívne volanie z riadka 21. Vo štvrtom sme vrchol 2 zmazali. Výsledné mapovanie je $4 \rightarrow C, 3 \rightarrow B, 2 \rightarrow \emptyset, 1 \rightarrow A$.

RTED: Robust Tree Edit Distance algoritmus

RTED budeme vnímať ako algoritmus na počítanie optimálnej stratégie - teda algoritmus, ktorý nám poradí ako najlepšie dekomponovať obidva stromy.

Funguje tak, že si predpočíta koľko podproblémov budeme musieť vyriešiť, ak použijeme stratégiu *left*, *right*, alebo *heavy*.

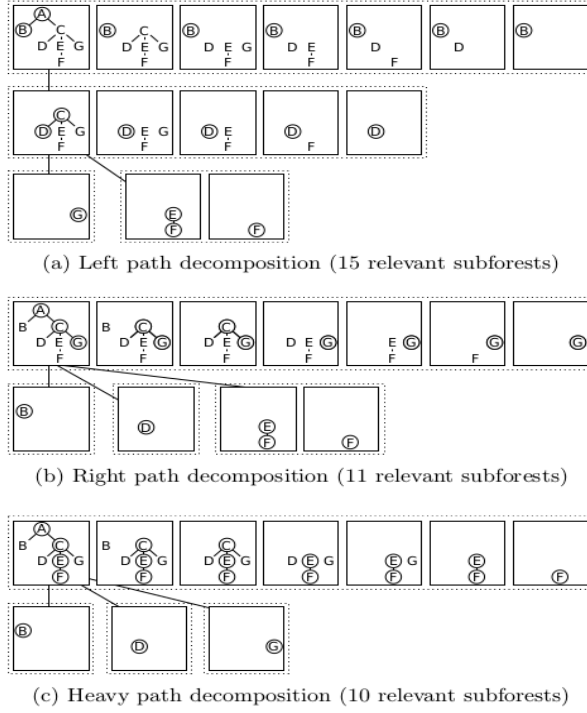
Algorithm 3 Mapping between trees

```

1: procedure GETMAPPING( $F, G, TreeDistance$ )
2:    $Mapping \leftarrow \emptyset$ 
3:    $\sigma \leftarrow$  random LRH strategy
4:    $\gamma \leftarrow$  path according to strategy  $\sigma$  (in  $F$  or  $G$ )
5:    $ForestDistance \leftarrow$  SINGLEPATH( $F, G, TreeDistance, \gamma$ )
6:   while  $F \neq \emptyset \wedge G \neq \emptyset$  do
7:      $v \leftarrow$  GETRELEVANTSUBFORESTNODE( $F, \sigma$ )
8:      $w \leftarrow$  GETRELEVANTSUBFORESTNODE( $G, \sigma$ )
9:     if  $ForestDistance[F, G] = ForestDistance[F - v, G] + c_{del}$  then
10:       $Mapping \leftarrow Mapping \cup (v \rightarrow 0)$ 
11:       $F \leftarrow F - v$ 
12:     else if  $ForestDistance[F, G] = ForestDistance[F, G - w] + c_{ins}$  then
13:       $Mapping \leftarrow Mapping \cup (0 \rightarrow w)$ 
14:       $G \leftarrow G - w$ 
15:     else
16:       if  $F$  and  $G$  are both trees then
17:          $Mapping \leftarrow Mapping \cup (v \rightarrow w)$ 
18:          $F \leftarrow F - v$ 
19:          $G \leftarrow G - w$ 
20:       else
21:          $Mapping \leftarrow Mapping \cup$  GETMAPPING( $F_v, G_w, TreeDistance$ )
22:          $F \leftarrow F - F_v$ 
23:          $G \leftarrow G - G_w$ 
24:   return  $Mapping$ 

25: procedure GETRELEVANTSUBFORESTNODE( $Forest, \sigma$ )
26:    $\gamma \leftarrow$  path in  $Forest$  according to strategy  $\sigma$ 
27:   if  $r_L(Forest) \in \gamma$  then
28:     return  $r_R(Forest)$ 
29:   else
30:     return  $r_L(Forest)$ 

```



Obr. 2.6: Celková dekompozícia pomocou LRH stratégií 15

Definícia 8. Celková dekompozícia lesa (full decomposition) F , $\mathcal{A}(F)$ je množina všetkých podlesov F , ktoré dostaneme rekurzívnym odstránením najľavejšieho alebo najpravejšieho koreňového vrcholu - $r_L(F)$ a $r_R(F)$ - z F a následne aj všetkých jeho podlesov:

$$\begin{aligned}\mathcal{A}(\emptyset) &= \emptyset \\ \mathcal{A}(F) &= F \cup \mathcal{A}(F - r_L(F)) \cup \mathcal{A}(F - r_R(F))\end{aligned}$$

Celková dekompozícia pomocou LRH stratégií je znázornena na obrázku 2.6. Vidíme, že najmenší počet relevant subforests má *heavy* dekompozícia (10).

Nasledujúca lemma, ktorú pre *heavy* cesty dokázali Demaine a kol. (2009) a pre *left* cesty zase Zhang a Shasha (1989)² nám ukáže, že počet problémov ktoré potrebujeme vyrátať závisí od výberu dekompozičnej stratégie.

Lemma 3. Počet podproblémov (relevant subproblems) počítaných *SinglePath* funkciou pre dvojicu stromov F a G je rovná

$$\# = \begin{cases} |F| \times |\mathcal{F}(G, \Gamma^L(G))| & \text{pre left cesty} \\ |F| \times |\mathcal{F}(G, \Gamma^R(G))| & \text{pre right cesty} \\ |F| \times |\mathcal{A}(G)| & \text{pre heavy cesty} \end{cases}$$

Lemma 4. Minimálny počet podproblémov (cena počítania danou stratégiou),

²Pre *right* cesty to platí obdobne, stačí nám zmeniť poradie medzi potomkami vrcholov

ktoré potrebujeme vyrátať pri použití GTEDu je

$$cena(F, G) = \min \left\{ \begin{array}{ll} |F| \times |\mathcal{A}(G)| & + \sum_{F' \in F - \gamma^H(F)} cena(F', G) \\ |G| \times |\mathcal{A}(F)| & + \sum_{G' \in G - \gamma^H(G)} cena(G', F) \\ |F| \times |\mathcal{F}(G, \Gamma^L(G))| & + \sum_{F' \in F - \gamma^L(F)} cena(F', G) \\ |G| \times |\mathcal{F}(F, \Gamma^L(F))| & + \sum_{G' \in G - \gamma^L(G)} cena(G', F) \\ |F| \times |\mathcal{F}(G, \Gamma^R(G))| & + \sum_{F' \in F - \gamma^R(F)} cena(F', G) \\ |G| \times |\mathcal{F}(F, \Gamma^R(F))| & + \sum_{G' \in G - \gamma^R(G)} cena(G', F) \end{array} \right.$$

pričom Γ^L a Γ^R sú funkcie vracajúce left a right cestu v lese.

Dôkaz. Je uvedený v 15

□

Popíšeme algoritmus 4 - RTED, od tvorcov Pawlik a Augsten (2011). Vďaka jeho rýchlosti $\mathcal{O}(n^2)$ si optimálnu stratégiu nájdeme a aj vypočítame GTED v čase $\mathcal{O}(n^3)$.

Algoritmus prechádza vrcholmi stromov a spočíta si, aká je optimálna dekompozičná stratégia pre danú dvojicu podstromov, teda pri akej potrebujem vyrátať minimálny počet podproblémov.

Stromom prechádza v postorder poradí, teda najprv navštívi potomkov zľava doprava, a až tak samotný vrchol. Toto poradie je zvolené kvôli tomu, aby sa znížila pamäťová náročnosť a nemuseli ukladať hodnoty medzi všetkými dvojicami podstromov. Namiesto toho inkrementujeme hodnotu v rodičovskom vrchole (procedúry *Update*) pri každej návšteve jeho potomka.

Lemma 5. *Algoritmus 4 vyráta optimálnu LRH stratégiu pre dvojicu podstromov F a G a časová náročnosť algoritmu je $\mathcal{O}(n^2)$.*

Dôkaz. Je uvedený v Pawlik a Augsten (2011).

□

Algorithm 4 Optimal strategy

```

1: procedure RTED( $F, G$ )
2:    $L_v, R_v, H_v \leftarrow$  arrays  $|F| \times |G|$ 
3:    $L_w, R_w, H_w \leftarrow$  arrays  $|G|$ 
4:   for all  $v$  postorder in  $F$  do
5:     for all  $w$  postorder in  $G$  do
6:       if  $v$  is leaf then
7:          $L_v[v, w] \leftarrow R_v[v, w] \leftarrow H_v[v, w] \leftarrow 0$ 
8:       if  $w$  is leaf then
9:          $L_w[w] \leftarrow R_w[w] \leftarrow H_w[w] \leftarrow 0$ 
10:       $C := \{$ 
11:         $(|F_v| \times \mathcal{A}(G_w) + H_v[v, w], \gamma^H(F)),$ 
12:         $(|G_w| \times \mathcal{A}(F_v) + H_w[w], \gamma^H(G)),$ 
13:         $(|F_v| \times |\mathcal{F}(G_w, \Gamma^L(G))| + L_v[v, w], \gamma^L(F)),$ 
14:         $(|G_w| \times |\mathcal{F}(F_v, \Gamma^L(F))| + L_w[w], \gamma^L(G)),$ 
15:         $(|F_v| \times |\mathcal{F}(G_w, \Gamma^R(G))| + R_v[v, w], \gamma^R(F)),$ 
16:         $(|G_w| \times |\mathcal{F}(F_v, \Gamma^R(F))| + R_w[w], \gamma^R(G))$ 
17:       $\}$ 
18:      Get  $(c_{min}, \gamma_{min}) \in C$  such that  $c_{min} = \min\{c' | (c', \gamma') \in C\}$ 
19:       $Strategies[v, w] := \gamma_{min}$ 
20:      if  $v$  is not root of tree then
21:        UPDATE( $L_v, v, w, c_{min}, \gamma^L(parent(v))$ )
22:        UPDATE( $R_v, v, w, c_{min}, \gamma^R(parent(v))$ )
23:        UPDATE( $H_v, v, w, c_{min}, \gamma^H(parent(v))$ )
24:      if  $w$  is not root of tree then
25:        UPDATE( $L_w, w, c_{min}, \gamma^L(parent(w))$ )
26:        UPDATE( $R_w, w, c_{min}, \gamma^R(parent(w))$ )
27:        UPDATE( $H_w, w, c_{min}, \gamma^H(parent(w))$ )
28:    return  $Strategies$ 

29: procedure UPDATE( $Table, v, w, c_{min}, \gamma$ )
30:    $Table[parent(v), w] \stackrel{+}{=} \begin{cases} Table[v, w] & \text{if } v \in \gamma \\ c_{min} & \text{otherwise} \end{cases}$ 

31: procedure UPDATE( $Table, w, c_{min}, \gamma$ )
32:    $Table[parent(w)] \stackrel{+}{=} \begin{cases} Table[w] & \text{if } w \in \gamma \\ c_{min} & \text{otherwise} \end{cases}$ 

```

3. Kreslenie molekuly

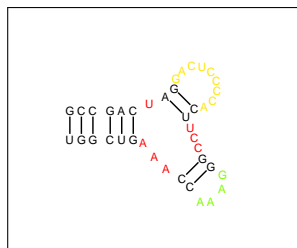
Po tom čo získame a aplikujeme mapovanie medzi šablonovou a cieľovou molekulou RNA, získame cieľovú molekulu s čiastočnou vizualizáciou, ktorej zvyšok treba dopočítať.

Po operáciach delete ostávajú v molekule prázdne diery, naopak po insertoch potrebujeme vypočítať, kam umiestnime bázy pár, resp. samotnú bázu, prípadne ešte potrebujeme pre ňu urobiť miesto. Update vrcholu v strome nerobí žiadne štrukturálne zmeny, zmení sa iba názov bázy na danom mieste.

Sekundárna štruktúra RNA obsahuje množstvo motivov popísaných na obrázku 1.4. Vo všeobecnosti ale sa každý z týchto motivov skladá zo stemu a loopu.

Stemom budeme ďalej nazývať časť RNA, ktorá zodpovedá vnútornému vrcholu v strome. Loop budeme označovať listy v RNA strome (lese), nezáleží či je to bulge, interior loop, hairpin alebo multibranch loop, ako aj ukazuje obrázok 3.1.

Stem začína vždy v najvyššom vrchole stromu (v smere ku koreňu), ktorý je zároveň vnútorným vrcholom a nemá žiadnych súrodencov, ktorý by boli rovnako vnútornými vrcholmi. To znamená, že do multibranch loop vchádza 1 stem (ten tu konci) a vychádza z nej niekoľko nových stemov. Naopak pre bulge a interior loop jeden stem vchádza do štruktúry ale pokračuje ďalej.



Obr. 3.1: Rozlisenie stemov a loopov v molekule: čierne su stemy, farebne odlisene su bazy patriace do jednej loopy

3.1 Štruktúry v RNA

V článku od Auber a kol. (2006) autori popisujú pravidlá vizualizácie sekundárnej štruktúry RNA.

Nakreslenie musí byť rovinné bez krížení, bázy tvoriace rôzne druhy loopov musia ležať na kružniciach a bázy tvoriace stem majú ležať na priamke. Ďalším pravidlom je, že vzdialenosť medzi bázami má byť konštantná, či už vzdialenosť medzi bázami jedného páru, alebo bázami sekvencie.

Ako je ukázane na obrázku ??, pravidla niesu niekedy rešpektované. To zťažuje použitie obrazka ako šablony, keďže vo výslednom obrázku chceme všetky tieto pravidlá rešpektovať.

3.2 Algoritmus

Čiastočnej vizualizácie, ktorú dostávame z mapovania sa chceme dotýkať čo najmenej. To znamená, že všetky zásahy sa snažíme robiť iba v miestach, ktoré boli dotknuté vkladáním alebo mazáním báz.

Jediné výnimky sú normalizácia vzdialenosti medzi bázovými párami a vyrovňovanie stémov.

3.2.1 Normalizácia vzdialeností v bázových pároch a vyrovňovanie stémov

Ako bolo uvedené, stémom rozumieme nevetviacu sa časť stromu tvorenú iba bázovými párami.

Algoritmus normalizácie vzdialeností medzi vrcholmi bázových párov stojí iba v preiterovaní celého stromu a ak nejaké párové vrcholy sú od seba príliš vzdialené, priblíži ich k sebe.

Vyrovňovací algoritmus prechádza všetky stemy. Z ich začiatkov vedie priamku, na ktorej majú byť podľa pravidiel uložené všetky stemové vrcholy. Rotáciami a posunutiami podstromov vieme docieľiť to, aby vrcholy stemu na tejto priamke ležali.

3.2.2 Operácie na stromoch

Čitateľa zoznámime s 2 operáciami, ktoré budeme vykonávať na molekule. Tie budeme používať nezávisle na tom, či vrcholy do stromu vkladáme alebo mažeme.

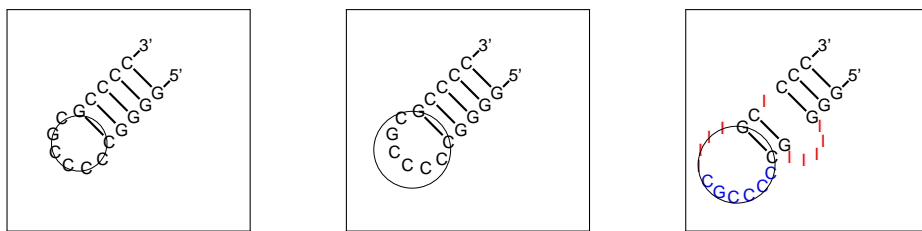
Algorithm 5 Rozloženie báz na kružnicu

```
1: procedure ROZLOZBAZY(Begin, End, Bases)
2:    $n \leftarrow$  veľkosť zoznamu báz Bases
3:    $\Gamma \leftarrow$  dostatočne veľká kružnica pre  $n$  bodov prechádzajúca bodmi Begin
   a End
4:    $\Pi \leftarrow$  rozdel kruhový oblúk kružnice  $\Gamma$  od Begin po End na  $n$  bodov
5:   for all  $i$  in  $1 \dots n$  do
6:     nastav pozíciu bázy Bases[ $i$ ] na bod  $\Pi[i]$ 
```

Algorithm 6 Posunutie podstromu

```
1: procedure POSUNPODSTROM(Root, Vector)
2:   for all vrchol  $V$  v podstrome vrcholu Root do
3:     if vrchol  $V$  už má určenú pozíciu, t.j. nieje práve vložený then
4:       pripočítaj k pozícii bázy  $V$  vektor Vector
```

Ako sme písali už skôr, všetky loop štruktúry majú byť uložené na kružniciach. K tomu nám pomôže funkcia `??`. Tá dostáva na vstupe zoznam báz *Bases* a dva body v rovine, *Begin* a *End*. Týmito bodmi potrebujeme viesť kružnicu, ktorá bude dostatočne veľká, teda aby na ňu všetky bázy zo zoznamu vošli. Veľkosťou kružnice v tomto prípade myslíme dĺžku kruhového obluku medzi vrcholmi *Begin* a *End*.



Obr. 3.2: Příklad zväčšovania kružnice a insertu do hairpinu

V našom programe používame iteračný algoritmus, ktorý ju pomaly zväčšuje alebo zmenšuje. Nakoniec buď nájde kružnicu, ktorej veľkosť je optimálna, alebo ani na maximálny počet krokov takú kružnicu nenájde a tak vráti tu z posledného kroku. Na obrázku 3.2 vidíme celý algoritmus zväčšovania kružnice.

Operácia v rámci algoritmu 6 nám pomôže urobiť miesto na novo vložené bázoové páry, alebo naopak ak sme niečo zmažali, tak dokáže celý podstrom priťahnúť späť.

3.2.3 Vkladanie nového vrcholu do stromu

Pri vkladaní nového vrcholu do stromu môžu nastať nasledovné možnosti.

Ak vkladáme list do hairpinu, je to jednoduché, potrebujeme iba použiť procedúru z algoritmu ?? s parametrami *Begin* = pozícia prvej bázy z bázoového páru, *End* = pozícia druhej bázy z páru a *Bases* = zoznam všetkých potomkov.

Trochu zložitejšie je to pri vkladaní listu do stemu. V tomto prípade buď už stem obsahoval nejaký loop, alebo vzniká nová. Najprv potrebujeme upraviť vzdialenosť medzi vrcholmi stemu, teda posunúť celý podstrom aby nám dané bázy vošli. To vyriešime algoritmom 6. Následne nájdeme kružnicu a bázy na ňu naukladáme.

Vkladanie bázoového páru do stemu je jednoduché. Najprv posunieme celý podstrom a urobíme tak miesto pre novú dvojicu báz, a potom ich uložíme na pozíciu kde by mala patriť. Môže sa stať, že vložením vrcholu do stemu zdedíme niekoľko listov z predka. V tomto prípade iba použijeme operáciu vloženia vrcholu a updatu loopov pred aj za vloženým vrcholom.

3.2.4 Modifikácia multibrach loop

Modifikácia multibrach loop je zložitejšia ako všetky predchádzajúce prípady. Obrázky sú väčšinou ručne upravené tak, aby bol čo najkompaktnejší a kvôli tomu sa často nerešpektujú pravidlá o kružnicovom tvare štruktúry. Kvôli tomu sa snažíme do tejto štruktúry nezasahovať, ak sa to dá.

Prekresleniu celej štruktúry sa môžeme vyhnúť napríklad pri zmene počtu listov medzi jednotlivými vetvami. Ak je zmena dostatočne malá, môžeme vrcholy rozťahnúť, alebo naopak priblížiť k sebe.

Ak sa jedná o pridanie/odoberanie celej vetvy stromu, modifikácií sa nevyhneme. V tom prípade potrebujeme rozdistribuovať všetky vrcholy patriace do loop na kružnicu. Je to podobný proces ako sa používa iba pre samotné loopy, ale potrebujeme posúvať celé podstromy a zrotovať ich správnym smerom.

3.2.5 Mazanie vrcholu zo stromu

Mazanie považujeme za inverznú operáciu voči vkladaniu do stromu. Vzhľadom k tomu, používame rovnaké operácie rozdistributiona vrcholov v loope, alebo posúvanie podstromu, ktoré sa deje v tomto prípade opačným smerom k predkovi.

4. TRAVeLer - Template RnA Visualization

Traveler je konzolová aplikácia programovaná v C++ a je určený pre operačné systémy UNIX-ového typu. Vyvíjaný a testovaný bol na Linux-e a FreeBSD. Podpora ostatných systémov nieje zaručená.

4.1 Inštalácia

Požadované programové vybavenie je:

- gcc verzie aspoň 4.9.2

Pri testovaní boli zaznamenané problémy s regulárnymi výrazmi, ktoré nám pomáhajú pri načítavaní vstupných súborov. Problém bol pri verzii gcc 4.7.2, ktorá plne nepodporovala potrebné výrazy.

Traveler preložíme zo zdrojových kódov postupnosťou príkazov z koreňového adresára:

- `cd src/`
- `make build`

Následne spustiteľný súbor je *src/build/traveler*.

4.2 Argumenty programu

Ak predpokladáme, že program leží na *PATH*, spúšťame ho nasledovne:

```
traveler [-h|--help]
traveler [OPTIONS] <TREES>
```

OPTIONS:

```
[-a|--all [--overlaps] [--colored] <FILE_OUT>]
[-t|--ted <FILE_MAPPING_OUT>]
[-d|--draw [--overlaps] [--colored] <FILE_MAPPING_IN> <FILE_OUT>]
[--debug]
```

TREES:

```
<-mt|--match-tree> FILE_FASTA
<-tt|--template-tree> [--type DOCUMENT_TYPE] DOCUMENT FILE_FASTA
```

Stručnú nápovedu k programu dostaneme štandardným *-h* alebo *--help* argumentom.

Prepínačmi *--ted* a *--draw* vieme oddeliť fázu počítania vzdialenosti pomocou TEDu a nasledného kreslenia.

Prepínač *--overlaps* po nakreslení obrázku v ňom vyznačí všetky miesta prekryvov, ak nejaké vznikli. Zároveň ich počet vypíše do samostatného súboru.

Následne rýchlejšie dokážeme identifikovať molekuly, ktoré potrebujú zvýšenú pozornosť.

Prepínač `--colored` aktivuje farebné zvýrazňovanie zmien v štruktúre stromu oproti šablone. Používame nasledovné kódovanie farbami:

- *Cervena* vložené bázy
- *Zelena* editované bázy
- *Modra* bázy ktoré sme potrebovali presunúť
- *Hneda* podstromy prekreslených multibranch loop

Farbami zvýrazňujeme zmeny v strome, to znamená, že ak sa bázo- vý pár zmení v jednej báze, celý bude označený ako editovaný.

Modrou označujeme časti, ktoré sme z nejakého dôvodu potrebovali presunúť a prekresliť. Typickým príkladom je prekreslenie loopy po vložení/zmazaní nejakej bázy. Vtedy sme vložili napríklad 1 bázu ale potrebovali presunúť ďalších 10 ktoré už v loop boli.

Hnedou farbou označujeme celé podstromy multibranch loopy, ktorú sme museli prekresliť. V týchto prípadoch vznikajú často veľké prekryvy a týmto ich odlišujeme od ostatných, nečakaných.

Prepínač `--match - tree` nám určuje RNA molekulu ktorú ideme vizualizovať, `--template - tree` šablónu. Strom vizualizovanej molekuly sa načítava iba z fasta súboru, kdežto pri šablónovej molekule potrebujeme aj jej obrázok. Viac informácii ohľadom parametra `--type` nájdete v kapitole ??.

4.2.1 Formát fasta súboru

Ako formát súborov kodujúcich stromy používame trochu upravený fasta formát.

Súbor na prvom riadku obsahuje názov molekuly hneď za znakom `>` až po prvú medzeru. Na ďalších riadkoch obsahuje znaky sekvencie RNA a znaky kodujúce sekundárnu štruktúru. Je zvykom, že riadky su široke najviac 80 znakov.

Fasta súbor pre šablonovú molekulu RNA potrebuje iba názov a zatvorenie, pre vizualizovanú molekulu aj sekvenciu. Je to dané tým, že sekvenciu si vieme vybrať z obrázka šablony.

4.3 Príklad vstupu

Teraz uvidíme príklad vstupu pre malú podjednotku ribozomálnej RNA myši, konkrétne príklad fasta súboru 4.1, podporovaného formátu post script súboru 4.2 a následne aj obrázok vizualizácie v post script súbore 4.3.

Poznámka. Podporujeme iba jeden formát PostScript súborov - ten používa databáza CRW publikovaná Cannone a kol. (2002). Ďalšie rozšírenia podpory iných formátov rozoberáme v kapitole ??.

Obr. 4.1: Príklad fasta súboru

Obr. 4.2: Príklad podporovaného formátu post script súboru

4.4 Výstupne súbory

Program generuje 2 druhy vystupov. Prvým je uloženie tabuľky mapovania TED algoritmu a druhým sú obrázky vo formáte SVG a PS.

Označme $T1$ strom šablony a $T2$ vizualizovaný strom.

Formát mapovacieho súboru je nasledovný:

Prvý riadok obsahuje *DISTANCE* : n , kde n je editačná vzdialenosť medzi $T1$ a $T2$.

Ostatné riadky sú vo formáte $i\ j$, kde $i, j \geq 0$. Inymi slovami:

- 1 2 - prvý vrchol z $T1$ potrebujeme namapovať na druhý vrchol z $T2$
- 0 2 - do výsledného stromu vkladáme druhý vrchol z $T2$
- 1 0 - zo stromu $T1$ mazeme prvý vrchol

PostScript súbor je zložený z hlavičky, v ktorej sú definície kresliacich funkcií za ktorými sú riadky kreslenia molekuly. Príklad je na obrázku 4.4.

Najprv definujeme operácie kreslenia v hlavičke súboru - *lwline*, *lwstring* a *lwarc* - kreslenie čiar, textu a kružníc. Za ktorými nasleduje samotné kreslenie molekuly.

Podobne funguje kreslenie v SVG súbore, ktorého príklad je na obrázku 4.5. Elementy *< text >* vypisujú na danú pozíciu text, *< line >* naopak kreslia čiary a *< circle >* zase kružnice.

4.5 Rozšírenie podpory iných vstupných obrázkov

Ako sme už uviedli, momentálne podporujeme iba jediný vstupný formát vstupných obrázkov. Je ním PostScript formát používaný databázou CRW od autorov Cannone a kol. (2002).

Definícia 9. *Extractor bude nejaký objekt, ktorý vie zo súboru určitého typu vynať potrebné položky reprezentujúce RNA sekvenciu a pozíciu báz na obrázku.*

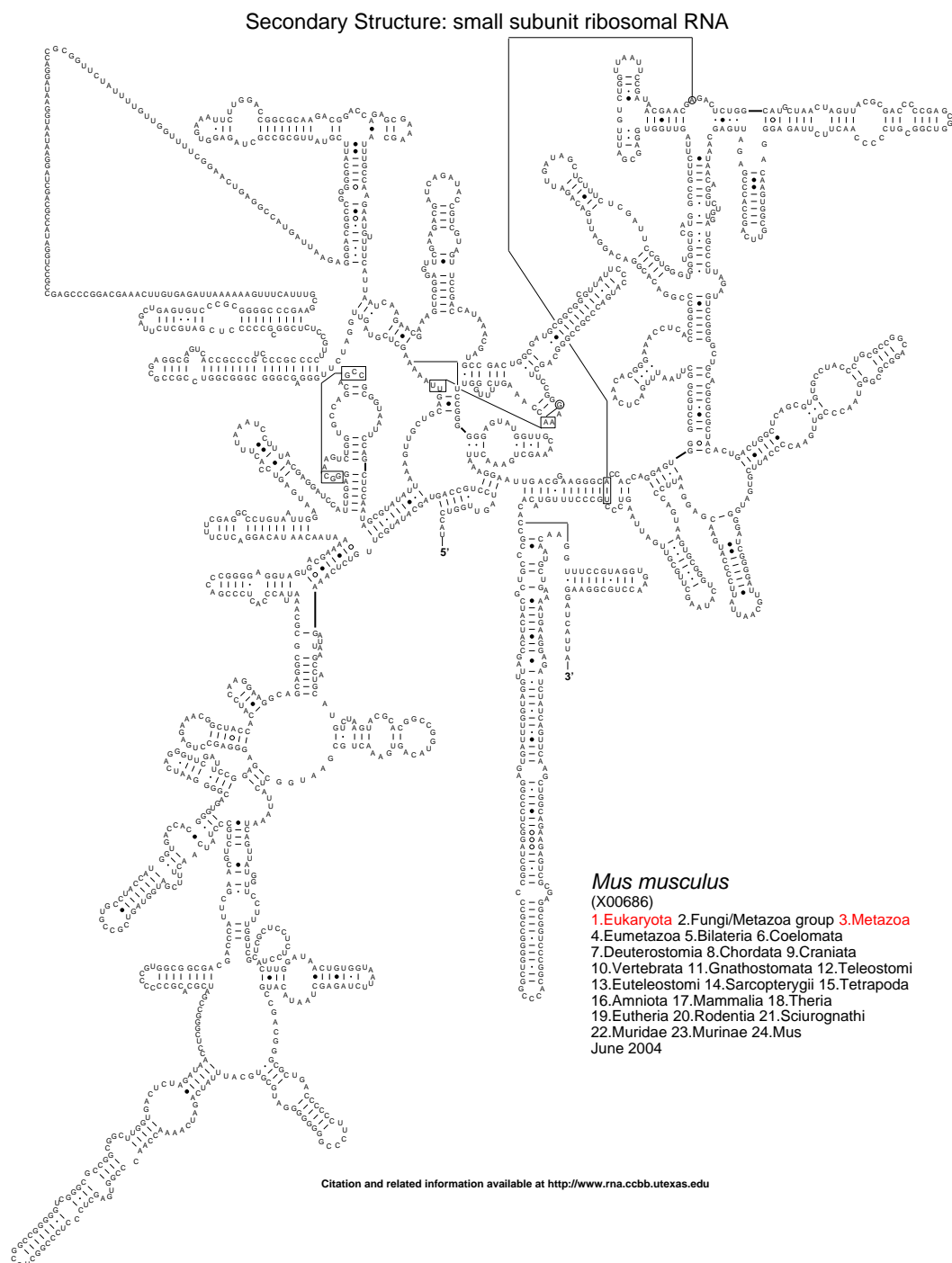
Pri tvorbe aplikácie sme už mysleli na budúcnosť a načítavanie súboru robíme v jednom ľahko rozširiteľnom module. Ten sa na základe typu v parametre *--template - tree* rozhoduje aký extractor použiť. Predvolený a jediný implementovaný je PostScript extractor fungujúci nad súbormi z CRW databázy.

Na implementovanie extractora potrebujeme implementovať existujúce rozhranie *extractor*, čo znamená implementovať metódu *init* s parametrom názvu súboru.

Jej úloha je zo súboru získať sekvenciu RNA a pozície báz.

Poslednou úlohou je pridať dvojicu (*nzov_extractor*, *extractor*) do tabuľky implementovaných v metóde *create_extractors()*.

Následným volaním *--template - tree --type nzov_extractor* začneme používať nás novo implementovaný extractor.



Obr. 4.3: Príklad vstupného obrázka

```

%!
/lwline {newpath moveto lineto stroke} def
/lwstring {moveto show} def
/lwarc {newpath gsave translate scale /rad exch def /ang1 exch def /ang2 exch def 0.0 0.0
  rad ang1 ang2 arc stroke grestore} def
/Helvetica findfont 8.00 scalefont setfont
0.36 0.46 scale
219.18 1384.80 translate
0          1          0          setrgbcolor
(5')       298.311     -268.09     lwstring
0          0          0          setrgbcolor
(U)        303.3       -273        lwstring
(A)        303.3       -265        lwstring
(C)        303.3       -257        lwstring
(C)        303.501     -248.682     lwstring
305.5      -241.809     302.5       -230.191     lwline
(U)        311.246     -246.682     lwstring
...
showpage

```

Obr. 4.4: Format vystupneho PostScript suboru

```

<svg
  xmlns="http://www.w3.org/2000/svg"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  width="1133.333333"
  height="1466.666667"
  viewBox="0 0 1139.172822px 1450.347571px"
  style="
    font-size: 8px;
    stroke: none;
    font-family: Helvetica; ">

  <text
    x="517.486977"
    y="603.524781"
    style="
      stroke: rgb(0, 255, 0); ">5'</text>

  <line
    x1="681.175823"
    y1="650.435118"
    x2="681.175823"
    y2="662.435118"
    style="
      stroke: rgb(0, 0, 0);
      stroke-width: 2; "/>

  <circle
    cx="616.350806"
    cy="427.616196"
    r="6.276645"
    style="
      stroke: rgb(0, 0, 0);
      fill: none; "/>

  ...
</svg>

```

Obr. 4.5: Formát vystupného SVG súboru

5. Výsledky práce

V tejto kapitole zhrnieme výsledky, ktoré naša práca dosiahla.

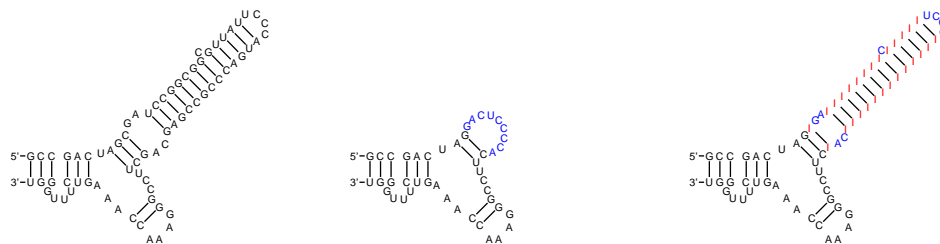
Už v predchádzajúcich kapitolách sme sa stretli s niekoľkými príkladmi, napríklad ako si program poradil s vkladaím do hairpinu - obrázok 3.2.

Na ďalšom (obrázok 5.1) simulujeme mazanie s následným vkladáním, teda 2 k sebe inverzne operácie. Po zmazení bázových párov na hornej vetve molekuly, sa nám všetky nepárové bázy ziali a vytvorili jednu loop. Následne po opätovnom vložení bázových parov (pre lepsie zviditelnenie sme ich označili Ľ”), vzniklá štruktúra veľmi podobná predchádzajúcej.

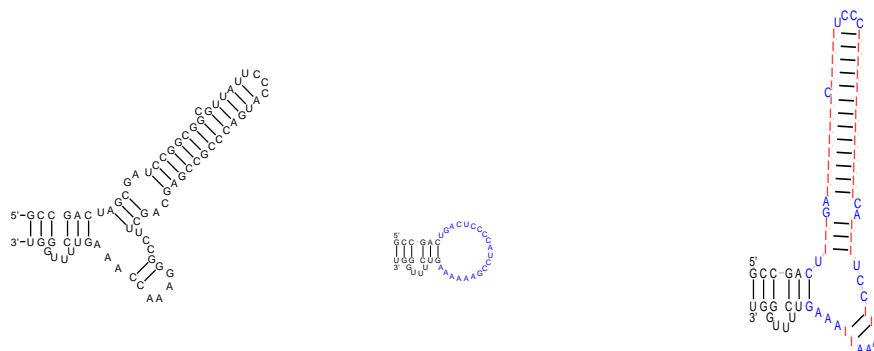
Obrázok má za cieľ ukázať, že vieme znovu nakresliť pôvodnú štruktúru iba s malými zmenami v pozícií nukleotidov (výsledne loopy sú trochu plytšie ako pôvodne).

Rovnako obrázok 5.2 rekonštruuje vetvenie sa stromu. Ako je vidieť, v tomto obrázku je už viac rozdielov, vychylenie je celkom badateľné.

Na takto malých častiach bez veľkých vetvení nam ani taketo zmeny nevadia. Pri veľkých molekulach ako ukazeme neskor problémy nastavajú.



Obr. 5.1: Inverzne operácie: rekonštrukcia stemu

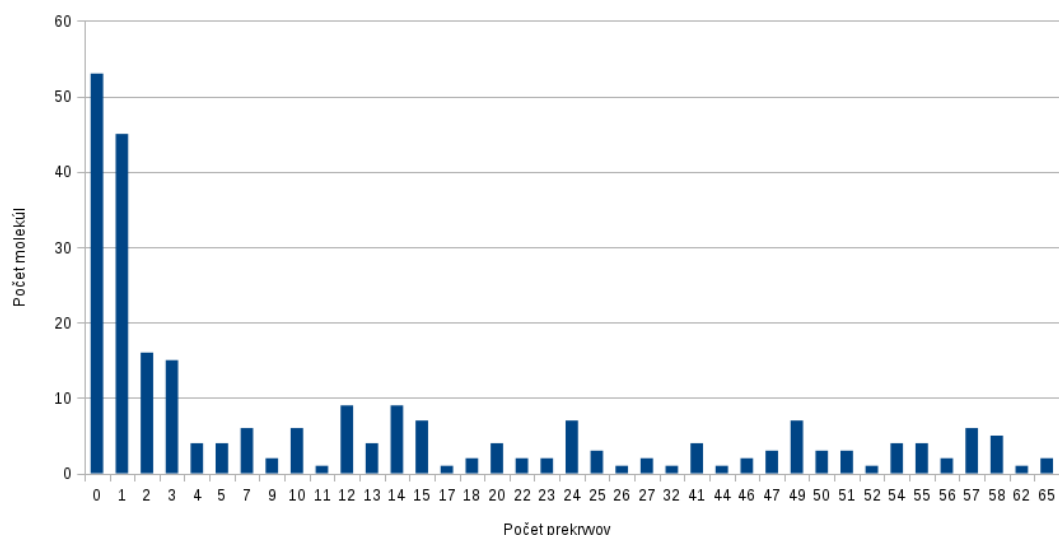


Obr. 5.2: Inverzné operácie: rekonštrukcia multibranch loop

Ako ďalšie, testovali sme schopnosť nášho algoritmu vizualizovať známu podjednotku 16S ribozomálnej RNA na živočísnej ríši. CRW databáza obsahuje 16 organizmov so známou sekundárnou štruktúrou.

Ribozomálna RNA bola vybrata lebo je v centre záujmu mnohých výskumov a taktiež kvôli jej veľkosti a zložitosti.

Náš vizualizačný test sme spustili na všetky páry RNA, z ktorých sme získali 256 vizualizácií.



Obr. 5.3: Počet prekryvov v testovaných molekulách

Na obrázku 5.3 vidíme počty molekúl s daným počtom prekryvov. Je ale niekoľko typov molekúl, u ktorých sme nejaké prekryvy čakali - napríklad, ak sme v nej potrebovali prekresliť multibranch loop. Takúto závislosť nám vyjadrujú ďalšie dva grafy, prvý - 5.5 nám ukazuje, že ak program musel rotovať a prekreslovať multibranch loopy, nedarilo sa mu najlepšie. Naopak, ak z prvého grafu odoberieme molekuly, ktoré museli použiť rotácie - graf ??, vidíme, že algoritmus šablonovej vizualizácie si viedol celkom dobre, prekryvy vznikali iba ojedinele.

Z tabuľky 5.1 je vidieť, že počet prekryvov závisí od počtu operácií vkladania a mazania ktoré v molekule musíme urobiť. Štatistika pracuje s prvou, piatou, desiatou a pätnástou najbližšou molekulou z pohľadu *tree – edit – distance*.

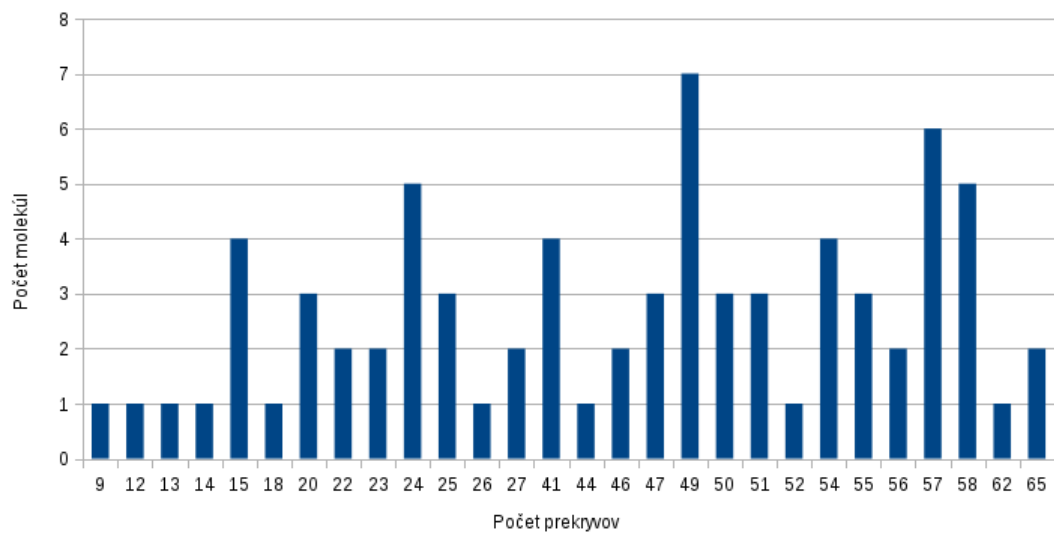
Zaujímavosťou je, že ako najvzdialenejšiu molekulu (v poradí pätnástu) si všetci vybrali molekulu od jedného konkrétneho zástupcu *echinococcus_granulosus* a vzdialenosť je 805,63 s odchylkou 12,92.

5.1 Celkové výsledky

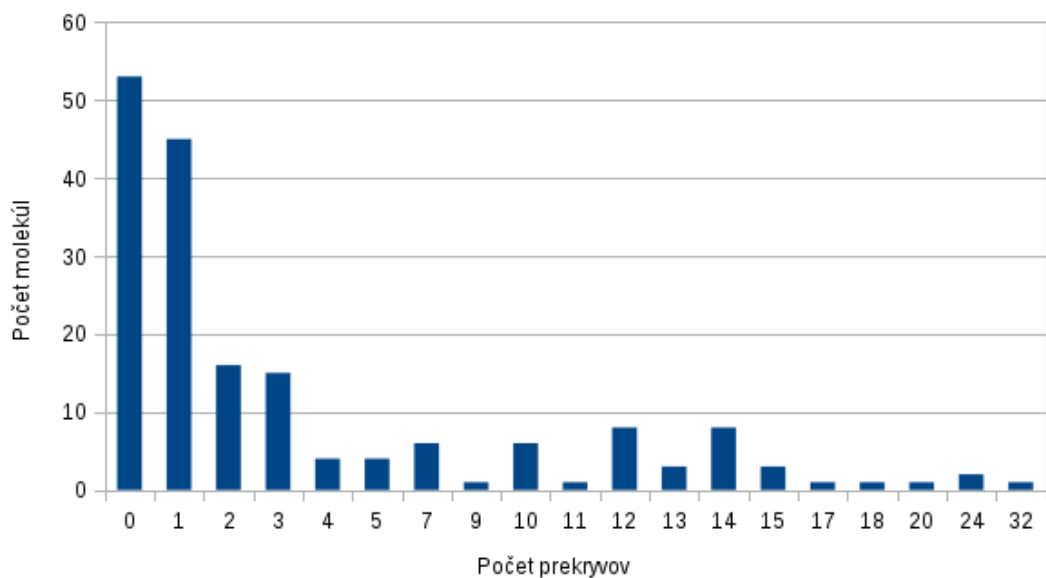
V tejto kapitole uvedieme vygenerované obrázky niektorých molekúl a na nich ukážeme časté problémy, ktoré pri vizualizácii nastávali.

Vzdialenosť	Počet prekryvov (priemer)	Smerodajná odchýlka
1.	5,13	1,64
5.	13,38	9,57
10.	14,13	12,47
15.	15,25	0,66

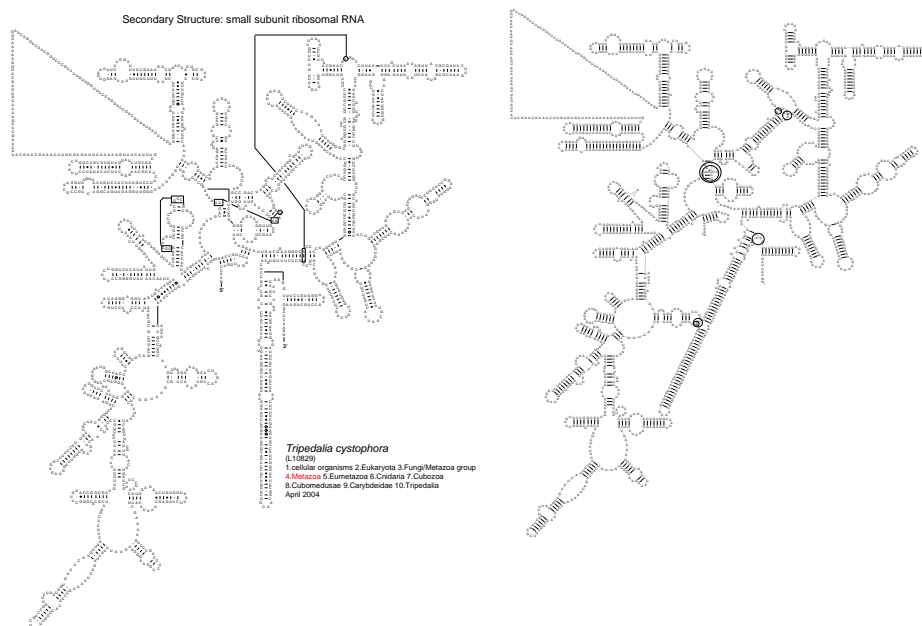
Tabuľka 5.1: Počty prekryvov v závislosti od tree-edit-distance vzdialenosti



Obr. 5.4: Počet prekryvov: molekuly, ktoré potrebovali prekreslit multibranch loop



Obr. 5.5: Počet prekryvov: molekuly bez prekreslovania multibranch loop



Obr. 5.6: Chyba pri otočení vetvy

5.1.1 Otáčanie vetvy kvôli existujúcej hrane

Jedným príkladom za všetky je molekula živočícha *Tripedaliacystophora* - meduzy. Po tom, čo sme dali molekulu nakresliť samu na seba vznikol problém, že celá jedna vetva molekuly sa otočila na jednu stranu. Je to spôsobené existenciou bázoového páru, ktorý je v pôvodnej molekule znázornený dlhšou lomenou čiarou.

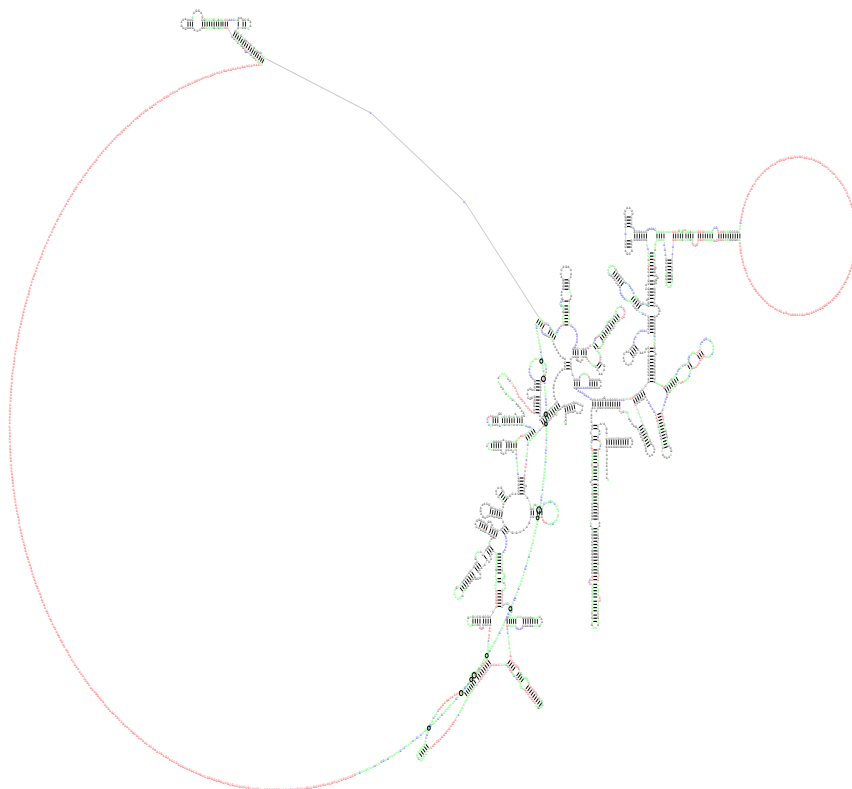
Keďže nás program všetky vzdialenosti normalizuje a následne ukladá bázy stemu na jednu priamku, vznikajú obrázky podobné 5.6.

5.1.2 Rozloženie báz na kružnicu

Niekedy sa prekresleniu celej loop nevyhneme. Ak napríklad vkladáme veľmi veľké množstvo báz na jedno miesto, dochádza k problémom načrtnutým na obrázku 5.7.

Na tomto konkrétnom príklade je nakreslený stem, vo vnútri ktorého je veľká loop. Kvôli tomu, že chceme dodržiavať pravidlá o kružnicovom tvare loopy, nájdeme kružnicu dostatočne veľkú. V tomto prípade až príliš veľkú.

Poznámka - vo vrchnej vetve ja taktiež znázornená kružnica, ale na rozdiel od spodnej obsahuje iba 2 vrcholy.



Obr. 5.7: Chyba pri rozkladaní báz na kružnicu

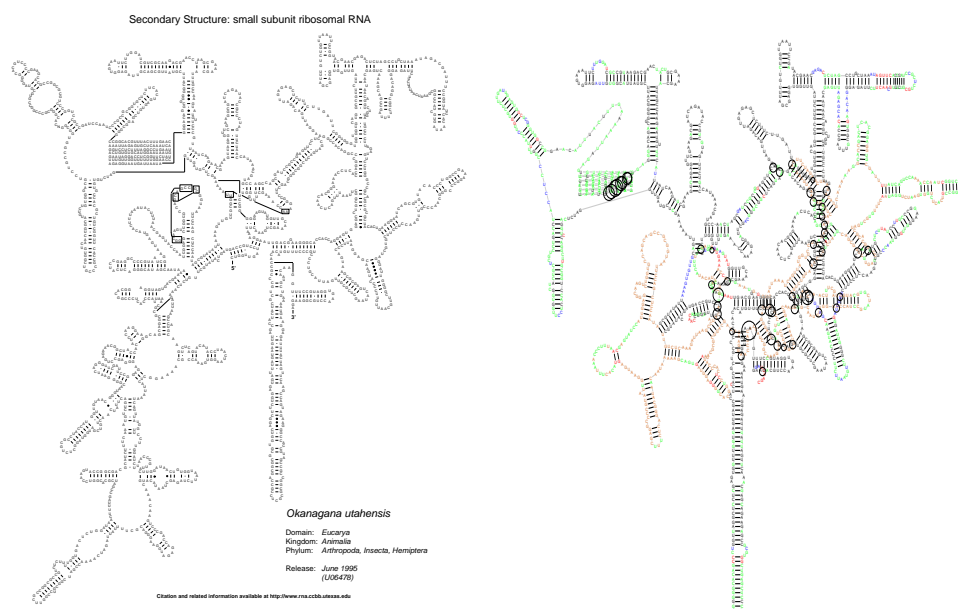
5.1.3 Otáčanie vetvy kvôli prekreslovaniu multibranch loopy

Ako už aj graf na obrázku ?? ukázal, prekresľovanie multibranch loopy a rotácie všetkých vetiev spôsobuje masívne prekryvy.

Pripájame jeden príklad na obrázku 5.8. Miesto vľavo dolu, kde začínajú bázy sa sfarbovať na hnedo, je multibranch loop, ktorú sme potrebovali z dôvodu vlozenej novej vetvy (označená červeno) prekresliť.

Výsledok je taký, že všetky vetvy sme uložili na kružnicu a pootáčali do vhodného smeru a tým vzniklo veľa prekryvov.

Na obrázku je vidieť ešte jednu vec, označené kríženia vo výslednom obrázku vľavo hore. V kapitole o úpravách multibranch loop sme spomínali, že prekresleniu celej loopy sa snažíme vyhnúť ak to ide. Predpokladáme, že ak je báz veľa a zmeny malé, bázy trochu poposúvame aby sa nový vrchol zmestil medzi ne, alebo práve naopak ich roztiahneme, aby sme vyplnili medzeru po starom. Kvôli tomu vyzerá táto štruktúra tak pomiešane a kvôli tomu na tomto mieste vznikajú ďalšie prekryvy.



Obr. 5.8: Chyba pri otáčani kvôli prekreslovaniu multibranch loopy

Závěr

Seznam použité literatury

- AUBER, D., DELEST, M., DOMENGER, J.-P. a DULUCQ, S. (2006). Efficient drawing of rna secondary structure. *Journal of Graph Algorithms and Applications*, **10**(2), 329–351. URL <http://eudml.org/doc/55423>.
- CANNONE, J., SUBRAMANIAN, S., SCHNARE, M., COLLETT, J., D'SOUZA, L., DU, Y., FENG, B., LIN, N., MADABUSI, L., MULLER, K., PANDE, N., SHANG, Z., YU, N. a GUTELL, R. (2002). The comparative RNA web (CRW) site: an online database of comparative sequence and structure information for ribosomal, intron, and other RNAs: Correction. *BMC Bioinformatics*, **3**(1), 15+. ISSN 1471-2105. doi: 10.1186/1471-2105-3-15. URL <http://dx.doi.org/10.1186/1471-2105-3-15>.
- CARTHEW, R. W. a SONTHEIMER, E. J. (2009). Origins and Mechanisms of miRNAs and siRNAs. *Cell*, **136**(4), 642–655.
- DARTY, K., DENISE, A. a PONTY, Y. (2009). VARNAs: Interactive drawing and editing of the RNA secondary structure. *Bioinformatics*, **25**(15), 1974–1975.
- DE RIJK, P., WUYTS, J. a DE WACHTER, R. (2003). RnaViz 2: an improved representation of RNA secondary structure. *Bioinformatics*, **19**(2), 299–300.
- DEMAINE, E. D., MOZES, S., ROSSMAN, B. a WEIMANN, O. (2009). An optimal decomposition algorithm for tree edit distance. *ACM Trans. Algorithms*, **6**(1), 2:1–2:19. ISSN 1549-6325. doi: 10.1145/1644015.1644017. URL <http://doi.acm.org/10.1145/1644015.1644017>.
- DULUCQ, S. a TOUZET, H. (2003). *Combinatorial Pattern Matching: 14th Annual Symposium, CPM 2003 Morelia, Michoacán, Mexico, June 25–27, 2003 Proceedings*, chapter Analysis of Tree Edit Distance Algorithms, pages 83–95. Springer Berlin Heidelberg, Berlin, Heidelberg. ISBN 978-3-540-44888-4. doi: 10.1007/3-540-44888-8_7. URL http://dx.doi.org/10.1007/3-540-44888-8_7.
- ELIÁŠ, R. a HOKZA, D. (2016). RNA Secondary Structure Visualization Using Tree Edit Distance. *International Journal of Bioscience, Biochemistry and Bioinformatics*, **6**(1), 9–17. doi: 10.17706/ijbbb.2016.6.1.9-17.
- HAN, K., LEE, Y. a KIM, W. (2002). PseudoViewer: automatic visualization of RNA pseudoknots. *Bioinformatics*, **18 Suppl 1**, S321–328.
- KISS, T. (2002). Small nucleolar RNAs: an abundant group of noncoding RNAs with diverse cellular functions. *Cell*, **109**(2), 145–148.
- KLEIN, P. N. (1998). Computing the edit-distance between unrooted ordered trees. In *Proceedings of the 6th Annual European Symposium on Algorithms, ESA '98*, pages 91–102, London, UK, UK, 1998. Springer-Verlag. ISBN 3-540-64848-8. URL <http://dl.acm.org/citation.cfm?id=647908.740125>.

- KLEINKAUF, R., HOUWAART, T., BACKOFEN, R. a MANN, M. (2015). antaRNA—Multi-objective inverse folding of pseudoknot RNA using ant-colony optimization. *BMC Bioinformatics*, **16**, 389.
- LEVENSHTAIN, V. I. (1966). Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*, **10**, 707.
- LORENZ, R., BERNHART, S. H., HONER ZU SIEDERDISSEN, C., TAHER, H., FLAMM, C., STADLER, P. F. a HOFACKER, I. L. (2011). ViennaRNA Package 2.0. *Algorithms Mol Biol*, **6**, 26.
- PAWLIK, M. a AUGSTEN, N. (2011). Rted: A robust algorithm for the tree edit distance. *Proc. VLDB Endow.*, **5**(4), 334–345. ISSN 2150-8097. doi: 10.14778/2095686.2095692. URL <http://dx.doi.org/10.14778/2095686.2095692>.
- PONTY, Y. a LECLERC, F. (2015). Drawing and editing the secondary structure(s) of RNA. *Methods Mol. Biol.*, **1269**, 63–100.
- PUTON, T., KOZŁOWSKI, L. P., ROTHER, K. M. a BUJNICKI, J. M. (2013). CompaRNA: a server for continuous benchmarking of automated methods for RNA secondary structure prediction. *Nucleic Acids Res.*, **41**(7), 4307–4323.
- PÁNEK, J., HAJIČ, J. a HOKSZA, D. (2014). Template-based prediction of ribosomal rna secondary structure. In *Bioinformatics and Biomedicine (BIBM), 2014 IEEE International Conference on*, pages 18–20. doi: 10.1109/BIBM.2014.6999394.
- SCOTT, J. (1988). Social network analysis. *Sociology*, **22**(1), 109–127. doi: 10.1177/0038038588022001007. URL <http://soc.sagepub.com/content/22/1/109.abstract>.
- TAI, K.-C. (1979). The tree-to-tree correction problem. *J. ACM*, **26**(3), 422–433. ISSN 0004-5411. doi: 10.1145/322139.322143. URL <http://doi.acm.org/10.1145/322139.322143>.
- TAMASSIA, R. (2007). *Handbook of Graph Drawing and Visualization (Discrete Mathematics and Its Applications)*. Chapman & Hall/CRC. ISBN 1584884126.
- WIESE, K. C., GLEN, E. a VASUDEVAN, A. (2005). jviz.rna -a java tool for rna secondary structure visualization. *IEEE Transactions on NanoBioscience*, **4**(3), 212–218. ISSN 1536-1241. doi: 10.1109/TNB.2005.853646.
- YANG, H., JOSSINET, F., LEONTIS, N., CHEN, L., WESTBROOK, J., BERMAN, H. a WESTHOF, E. (2003). Tools for the automatic identification and classification of RNA base pairs. *Nucleic Acids Res.*, **31**(13), 3450–3460.
- ZHANG, K. a SHASHA, D. (1989). Simple fast algorithms for the editing distance between trees and related problems. *SIAM Journal on Computing*, **18**(6), 1245 – 1262.
- ZUKER, M. (2003). Mfold web server for nucleic acid folding and hybridization prediction. *Nucleic Acids Res.*, **31**(13), 3406–3415.

Zoznam obrázkov

1.1	Spojnicový graf	6
1.2	Kruhový graf	6
1.3	Vizualizácia sekundárnej štruktúry RNA človeka z CRW databázy 2	7
1.4	Štrukturálne motívy v RNA	8
1.5	Typy pseudouzlov podľa 12	8
1.6	Malá podjednotka vygenerovaná programom jViz.Rna 22	9
1.7	Malá podjednotka vygenerovaná programom Mfold 25	10
1.8	Malá podjednotka vygenerovaná programom RNAfold 14	11
1.9	Varianty reprezentácie vrcholov podľa 1	12
1.10	Molekula RNA a jej stromová reprezentácia	12
1.11	Príklad stromu. Je v ňom červene vyznačená <i>heavy</i> cesta	13
2.1	Ukážky TED operácií	14
2.2	Rekurzívny vzorec pre výpočet tree-edit-distance od Demaine a kol. (2009) a Pawlik a Augsten (2011)	15
2.3	Príklad výpočtu GTED medzi stromami F a G	19
2.4	Výsledná tabuľka <i>TreeDistance</i> medzi stromami z 2.3	20
2.5	Výpočet mapovania medzi stromami	20
2.6	Celková dekompozícia pomocou LRH stratégií 15	22
3.1	Rozlisenie stemov a loopov v molekule: čierne su stemy, farebne odlisené su bazy patriace do jednej loopy	25
3.2	Príklad zvacsovania kružnice a insertu do hairpinu	27
4.1	Príklad fasta súboru	31
4.2	Príklad podporovaného formátu post script súboru	31
4.3	Príklad vstupného obrázka	33
4.4	Format vystupného PostScript suboru	34
4.5	Formát vystupného SVG súboru	34
5.1	Inverzne operácie: rekonštrukcia stemu	35
5.2	Inverzné operácie: rekonštrukcia multibranch loop	35
5.3	Počet prekryvov v testovaných molekulách	36
5.4	Počet prekryvov: molekuly, ktoré potrebovali prekreslit multib- ranch loop	37
5.5	Počet prekryvov: molekuly bez prekreslovania multibranch loop .	37
5.6	Chyba pri otočení vetvy	38
5.7	Chyba pri rozkladaní báz na kružnicu	39
5.8	Chyba pri otáčani kvôli prekreslovaniu multibranch loopy	40

Zoznam tabuliek

5.1	Počty prekryvov v závislosti od tree-edit-distance vzdialenosti . . .	36
-----	---	----

Seznam použitých zkratek

Přílohy