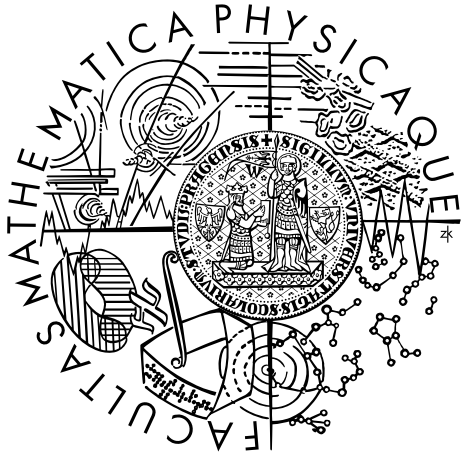
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

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 with a homologous structure with a good visualization is already available. [ii]

Keywords: RNA secondary structure, visualization, homology

Poděkování.

Obsah

Úvod

Následuje několik ukázkových kapitol, které doporučují, jak by se měla bakalářská práce sázet. Primárně popisují použití TEXové šablony, ale obecné rady poslouží dobře i uživatelům jiných systémů.

1. Uvod do študia štruktúry RNA a grafov

Na začiatku práce stručne zoznámime čitateľa s pojmamy, ktoré s RNA a jej štruktúrou súvisia.

1.1 Co je RNA

Ribonukleónová kyselina je nukleonová kyselina tvorená jedným vláknom kovalentne naviazaných ribonukleotidov, ktoré sú základnými stavebnými jendotkami nukleonových kyselín. Je biochemicky odlišná od DNA kôli prítomnosti hydroxilovej skupiny pripojenej ku každej molekule pentózy v reťazci. v DNA a RNA sa vyskytuje niekoľko variant nukleotidov (báz). U RNA sú to adeín (A), guanín (G), cytozín (C), uracyl (U), pri DNA sa namiesto uracylu vyskytuje tymin (T). Medzi jednotlivými bázami sa môžu vyskytovať vodíkové väzby. Nukleotidy majú vzájomnú preferenciu, čo znamená, že bázy vznikajú najčastejšie medzi A-U a C-G u RNA a podobne A-T a C-G u DNA.

Štrukturu nukleových kyselín môžeme chápat podľa stupňa zjednodušenia

- Primárna štruktúra - je určená poradím jednotlivých nukleotidov do polynukleotidového reťazca
- Sekundárna štruktúra - je daná parovaním medzi bázami molekuly
- Terciárna štruktúra - priestorové usporiadanie molekuly

DNA je dvojvláknova molekula, u ktorej spojenie medzi vlaknami sa realizuje na princípe komplementarity. Naopak, RNA je iba jednovláknova molekula a v snahe minimalizovať voľnu energiu molekuly, sa paruje sama na seba. V tomto hrajú rolu pritazlive sily medzi bázami.

V praci budeme štruktúrou myslieť práve sekundárnu štruktuúru RNA, ak nebude povedané inak.

Až donedávna sa myslelo, že funkcia RNA je obmedzená na prenos genetickej informácie z DNA v jadre bunky do ribozomu. Napríklad pri tvorbe bielkovín (mRNA), alebo transporter aminokyselin v ribozome bunky (tRNA). Avšak existuje mnoho ďalších, od relatívne malých molekúl tvorených desiatkami báz, ktoré pomáhajú pri expresii genov (miRNA, siRNA, tmRNA a dalsie), až po veľké, tvorené tisíckami nukleotidov (rRNA).

1.2 Sekundarna struktura rRNA + konzervovanost

Ako hlavný objekt záujmu sme si spomedzi všetkych druhov RNA vybrali práve ribozomálnu, najmä kvôli jej veľkosti a tomu, že existujúcim nástrojom práve veľkosť robí najväčšie problémy pri vizualizácii.

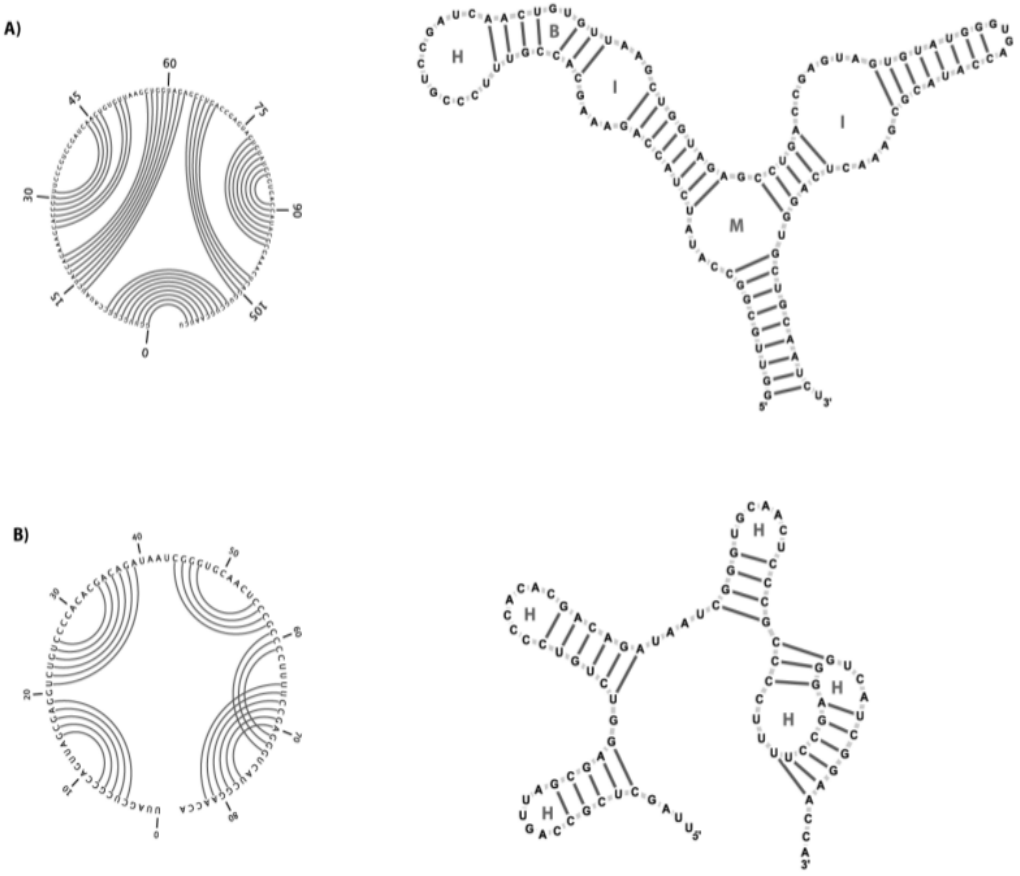
Definícia 1 (Primárna štruktra RNA)**.** *Nech* Σ *je abeceda* $\{A, C, G, U\}$*. Potom slovo* $W \in \Sigma^n$ *nad touto abecedou je sekvencia nukleotidov (baz) RNA.*

Jednotlivé nukleotidy sekvencie RNA budeme, ak bude zretelné o čo sa jedná, označovat priamo poradovým číslom, teda i bude označovať nukleotid W_i, resp. $W[i]$.

Definícia 2 (Sekundarna struktura RNA)**.** *Nech* W *je sekvencia podla definicie 1 dlzky n. Sekundarnou strukturou oznacime mnozinu* $\mathbb{S}$ *parov nukleotidov* (i, j) *takych, ze pre dva pary* (i, j) *a* $(k, l) \in \mathbb{S}$ *(bez ujmy na obecnosti* $i \leq k$*) plati jedno z nasledujucich:*

- $i = k \iff j = l$
- $i < j < k < l$*, cize par* (i, j) *predchadza par* (k, l)
- $i < k < l < j$*, cize par* (i, j) *obsahuje par* (k, l)

Prvá podmienka zabezpečuje, že nukleotid je najviac v jednom bazickom páre, druhá a tretia hovoria o usporiadani párov, buď sú na sebe nezávisle alebo na seba nadväzujú. Posledna podmienka zakazuje existenciu pseudouzlov (pseudoknots). Pseudouzol patrí medzi najčastejšie typy priestorového usporiadania RNA. Vytvára niekolko interakcii vrámci jednej molekuly a smyčky typu loops, ktore vznikajú medzi rôznimy molekulami.

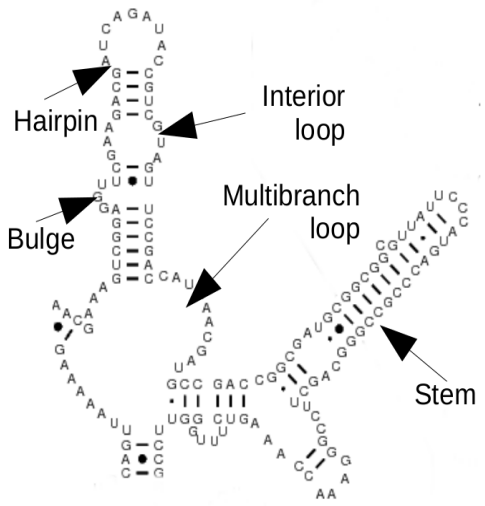


Obr. 1.1: Circular Feynman - kruhova reprezentacia sekundarnej struktury

1.2.1 Motivy

Motivom v RNA máme na mysli časti molekuly, ktoré vytvárajú určité štruktúry. Na obrazku 1.2 vidime motivy, ktore sa mozu v RNA vyskytovat.

Stem (stonka) je časť molekuly kde sa na seba paruju dva súvisle časti RNA vlákna. Interior loop spája dva stemy a medzi nimi na oboch stranách obsahuje nespárované bázy. Podobna je bulge (vypuklina), ale nespárované nukleotidy ma iba z jednej strany. Hairpin je medzi časťami vlákna, ktoré sa parujú sami na seba. Multibranch loop je podobná ako interior loop, ale spája dokopy viac stemov. V ďalšom rozprávani ámam bude stačiť rozdelenie na stem a loop.



Obr. 1.2: Strukturalne motivy v RNA

1.3 Grafové pojmy

Potrebujeme si definovať značenie a pojmy, ktoré budeme používať naprieč celou prácou. Z väčšej časti použijeme značenie od Pawlik a Augsten (2011).

1.3.1 Značenie

Definícia 3. *Usporiadaný zakorenený strom je orientovaný graf, v ktorom platí, že v ňom neexistujú cykly a že hrany sú orientované vždy v smere z predka na potomka. Okrem koreňa má každý vrchol svojho predka. Naviac tu 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 $\emptyset$.

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 plati 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 iba potomkovia v.
$F - v$ 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. *Nech F je strom, u a v jeho dva rôzne vrcholy. Hovoríme, že u je predkom 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.*

1.4 Stromová reprezentacia sekundarnej štruktury

Definicia 2 nám ponúka reprezentovať sekundárnu štrukturu ako usporiadaný strom.

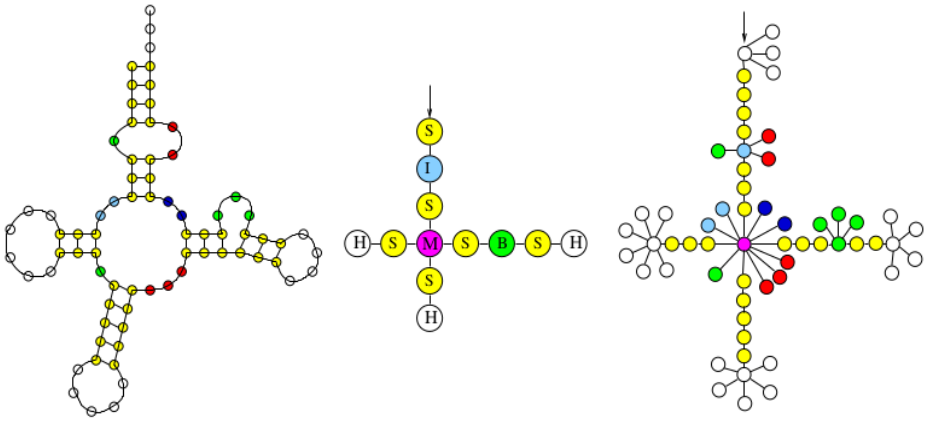


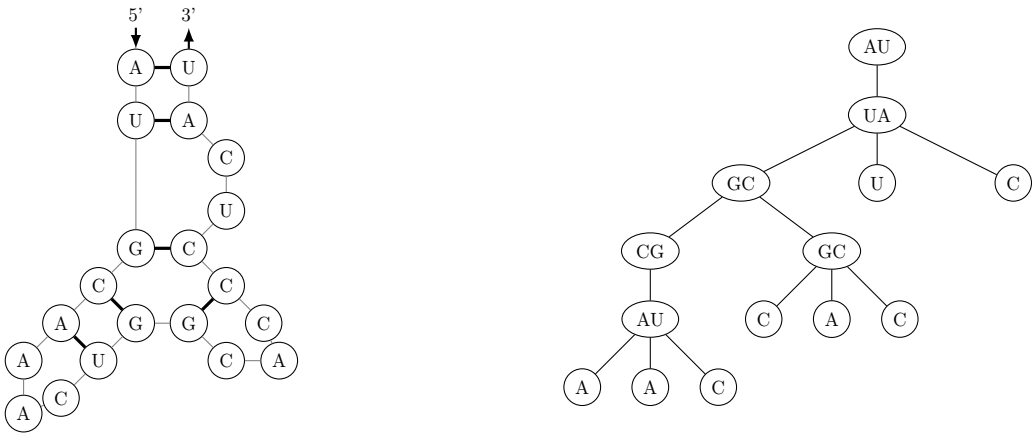
Figure 3: A secondary structure and its tree representations.

Obr. 1.3: Varianty reprezentacie vrcholov

Bez ujmy na obecnosti budeme o RNA hovoriť ako o strome, aj keď sa môže stať, že štruktúra nebude celistvá (teda nieje to strom, ale les). V tom prípade ale iba pripojíme koreňový vrchol, ktorého potomkovia budú dané stromy.

Každý vrchol stromu môže reprezentovať napríklad motiv v štruktúre RNA, nukleotid, alebo bázovy pár Príklady možno vidiet na obrázku 1.3.

V našej práci vrchol stromu reprezentuje bázovy pár (vnútorný vrchol) a nespárovanú bázu (list stromu). Štruktúru, do ktorej patrí si totiž vieme ľahko zistiť z potomkov vrcholu.

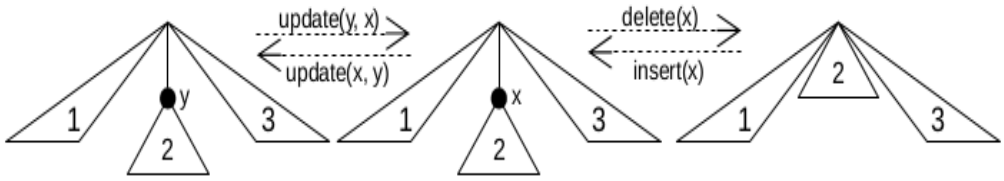


2. Tree-edit-distance algoritmus

Jadro aplikacie leží v použití tree-edit-distance (TED) algoritmu, vďaka ktorému dostaneme mapovanie medzi 2 RNA stromami. Mapovanie nám ukáže spoločné časti oboch RNA stromov. TED algoritmus je obdoba Levenstheinoveho string-edit-distance algoritmu. Problém u reťazcov je špecialným prípadom TED-u, kedy stromy zdegenerovali na cesty (spojový zoznam).

2.1 Hlavná myšlienka TED-u

Základ TED algoritmu je v rekurzivnom vzorci 2.2 z Demaine a kol. (2009) a Pawlik a Augsten (2011). Vzdialenosť medzi lesmi F a G, $\delta(F, G)$ je definovana ako minimálny počet editačných operácií, ktoré z F urobia G. Používame štandardne editačné operácie - delete, insert, update.



Obr. 2.1: Ukazky TED operacii

Delete, zmazanie vrcholu, znamená pripojiť k predkovi všetkých jeho potomkov so zachovanim poradia medzi nimi. Insert, vložnie vrcholu, je opačná operácia k delete, čo znamená, že vkladáme vrchol medzi rodiča a nejakých jeho, po sebe nasledujúcich potomkov. Update iba zmeni hodnotu vo vrchole stromu.

2.2 Značenie

V tejto kapitole sa budeme riadiť značením Pawlik a Augsten (2011). Teda, používame definiciu stromu a lesa z 3. Ak F je les (strom), N_F označuje množinu jeho vrcholov a E_F množinu jeho hrán. Platí ďalej že $E_F \subseteq N_F \times N_F$. $\emptyset$ označuje prázdny strom, resp. prázdny les. Podles lesa F je graf $\tilde{F}$ s vrcholmi $N_{\tilde{F}} \subseteq N_F$ a hranami $E_{\tilde{F}} \subseteq E_F \cap N_{\tilde{F}} \times N_{\tilde{F}}$. Obdobne to platí aj pre podstrom stromu T. F_v označuje podstrom F zakorenený vo v, t.j. v strome ostávajú iba potomkovia v. $F - v$ budeme značiť les, ktorý dostaneme zmazaním vrcholu v z F, spolu so všetkými hranami zasahujúcimi do v. Podobne $F - F_v$ budeme značiť les, ktorý dostaneme zmazaním podstromu F_v z F.

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

Vo vzorci 2.2 počítame editačnú vzdialenosť $\delta(F, G)$, c_{del}, c_{ins}a c_{upd}sú ceny zmazania, vloženia a editácie vrcholu v strome a r_F a r_G sú korene, buď obidva

najpravejšie alebo najľavejšie (tzn. vyberieme najpravejší/najľavejší strom lesa a jeho koreň).

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

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

Obr. 2.2: Rekurzívny vzorec pre výpočet tree-edit-distance

2.3 Algoritmy dynamického programovania

Tai (1979) predstavil algoritmus s priestorovou a časovou zložitosťou $\mathcal{O}(m^3 \cdot n^3)$, Zhang a Shasha (1989) algoritmus následne vylepšili pozorovaním toho, že nepotrebujeme vzdialenosti medzi všetkými pármi podlesov. Algoritmus mal časovú zložitosť $\mathcal{O}(m^2 \cdot n^2)$ a priestorovú $\mathcal{O}(m \cdot n)$. Klein (1998) dosiahol časovú zložitosť $\mathcal{O}(m^2 \cdot n \cdot \log n)$, avšak jeho riešenie potrebovalo 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) predviedli worst-case optimálny algoritmus pre tree-edit-distance. Jeho časová a priestorová zložitosť je $\mathcal{O}(m^2 \cdot n \cdot (1 + \log \frac{n}{m}))$ a $\mathcal{O}(m \cdot n)$. Pawlik a Augsten (2011) ukázali spojitosť medzi efektivnosťou predchádzajúcich algoritmov a tvarom stromov. Zovšeobecnili predchádzajúce prístupy a vytvorili algoritmus bežiaci vo worst-case čase $\mathcal{O}(m^3)$ a priestore $\mathcal{O}(m \cdot n)$. Ich algoritmus je teda efektívny pre všetky tvary stromov a nikdy nespadne do worst-case, ak existuje lepší smer výpočtu.

2.3.1 RTED

Ďalej sa v našej práci 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 definicia 6) a GTED (General Tree Edit Distance) algoritmus samotný výpočet rekurzie 2.2 s aplikovaním danej stratégie.

Definícia 6 (Dekompozičná stratégia)**.** *Nech F a G sú lesy. Dekompozičná stratégia v rekurzii 2.2 priradí každej dvojici podstromov F_v a G_w lesov F a G 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 na ceste z koreňa do listu. Najťažší vrchol je taký, v ktorého podstrome je najviac vrcholov.*

GTED: General Tree Edit Distance algoritmu

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.

Algorithm 1 General Tree Edit Distance for LRH strategies

```
1:  procedure GTED(F, G, TreeDistance, S)
2:      σ ← S[F, G]
3:      if σ ∈ σ*(F) then
4:          for all F' ∈ F − σ do
5:              TreeDistance ← TreeDistance ∪ GTED(F', G, TreeDistance, S)
6:          TreeDistance ← TreeDistance∪
7:              COMPUTE DISTANCE(F, G, TreeDistance, σ)
8:      else
9:          TreeDistance ← TreeDistance ∪ (GTED(G, F, TreeDistance^T, S^T))^T
10:     return TreeDistance
```

Poznámka. Funkcia $GetOrderedSubforests()$ v algoritme 2 vracia lesy zoradené v opačnom poradí, ako ich pridávame v definícii 7.

Algoritmus 1 funguje v troch krokoch.

Najprv podľa stratégie dekomponuje jeden zo stromov podľa cesty γ, bez ujmy na obecnosti, nech je to F a rekurzívne spočíta editačnú vzdialenosť medzi všetkými podstromami, ktoré susedia s dekompozicnou cestou a stromom G.

Následne pre všetky relevant-subtrees (viz definice 7) podstromy G' stromu G vyráta vzdialenosti medzi F_v a G' pomocou single-path funkcie. Tá dopočíta vzdialenosti medzi vrcholmi $v \in \gamma_F$ a stromami G'.

Definícia 7. *Relevant subtrees stromu F pre root-leaf cestu γ sú definované ako $F - \gamma$. Relevant subforests stromu F pre nejakú root-leaf cestu γ sú definované rekurzívne ako*

$$\mathcal{F}(\emptyset, \gamma) = \emptyset$$

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

Lemma 1. *Ak compute-distance 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 vyrátani 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 doslednému usporiadaniu lesov si v každom kroku pripravíme potrebne data pre ďalší krok algoritmu 2.

Algorithm 2 Single path function

```
1: procedure COMPUTE DISTANCE(F, G, TreeDistance, σ)
2:     if σ ∈ σ*(F) then
3:         for all G' ∈ RELEVANT SUBTREES(G) do
4:             SINGLE PATH(F, G', TreeDistance, σ)
5:     else
6:         for all F' ∈ RELEVANT SUBTREES(F) do
7:             SINGLE PATH(F, G, TreeDistance, σ)
8:
9: procedure SINGLE PATH(F, G, TreeDistance, σ)
10:    ForestDistance ← empty array |F| + 1 × |G| + 1
11:    ForestDistance[∅][∅] := 0
12:    for F' subforest in GET ORDERED SUBFORESTS(F, σ) do
13:        Last_F ← last added node to F'
14:        ForestDistance[F'][∅] := ForestDistance[F' − Last_F][∅]+
15:            C_del(Last_F)
16:    for G' subforest in GET ORDERED SUBFORESTS(G, σ) do
17:        Last_G ← last added node to G'
18:        ForestDistance[∅][G'] := ForestDistance[∅][G' − Last_G]+
19:            C_ins(Last_G)
20:    for F' subforest in GET ORDERED SUBFORESTS(F, σ) do
21:        for G' subforest in GET ORDERED SUBFORESTS(G, σ) do
22:            Last_F ← last added node to F'
23:            Last_G ← last added node to G'
24:            if both F' and G' are trees then
25:                C_min := min{
26:                    ForestDistance[F' − Last_F][G']+
27:                        C_del(Last_F),
28:                    ForestDistance[F'][G' − Last_G]+
29:                        C_ins(Last_G),
30:                    ForestDistance[F' − Last_F][G' − Last_G]+
31:                        C_upd(Last_F, Last_G)
32:                ForestDistance[F', G'] := C_min
33:                TreeDistance[Last_F][Last_G] := C_min
34:            else
35:                C_min := min{
36:                    ForestDistance[F' − Last_F)][G']+
37:                        C_del(Last_F),
38:                    ForestDistance[F'][G' − Last_G]+
39:                        C_ins(Last_G),
40:                    ForestDistance[F' − F_{Last_F}][G' − G_{Last_G}]+
41:                        TreeDistance[F_{Last_F}][G_{Last_G}]}
42:                ForestDistance[F'][G'] := C_min
```

Najprv si ešte ale vysvetlíme hodnoty používané v algoritme 2 v podmienkách na riadkoch 24 a 34. Prvé dva sú v oboch rovnaké. Počítame hodnotu zmazania vrcholu z F, resp. vloženia vrcholu do F.

Tretia hodnota sa líši podľa toho, či sú lesy zároveň aj stromami. Ak sú, tak na danom mieste je cena namapovania podstromov $F_v - v$ na $F_w - w$ a updatu vrcholu v na w. Inac, ked aspon jeden z lesov nieje stromom, tak cenu medzi F_{Last_F} a G_{Last_G} mame vyrátanú z predchádzajúcich krokoch, alebo z inej vetvy rekurzie.

Potom nastavíme hodnotu vzdialenosti medzi lesmi na minimum a v prípade že sú to obidva stromy, tak nastavíme aj ich vzdialenosť.

Najprv ešte ukážeme, že SPF používa vždy inicializované hodnoty a každú hodnotu nastavuje práve raz.

Poznámka. Nikdy nepoužívam 2x rovnakú cestu γ v strome. To vyplýva z toho, že po dekompozícií stromu podľa γ, cesta v ostatných stromoch neexistuje. 1

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, takze nikdy sa nedostanem do menšieho aby som mohol mu znovu nastaviť hodnotu. To iste plati aj pre $ForestDistance$.

□

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

Dôkaz. Hodnota $ForestDistance$ pre použitie s prázdnym lesom je inicializovaná, a pri každej iteracií algoritmu čítam iba z hodnôt z predchadzajúcich iteracií, 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 dekompozicií F už skôr. Naopak ak $Last_F$ leži na ceste, potom $Last_G$ je mimo cesty, a editačnú vzdialenosť sme vyrátali pri počítani relevant-subtrees.

□

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.

□

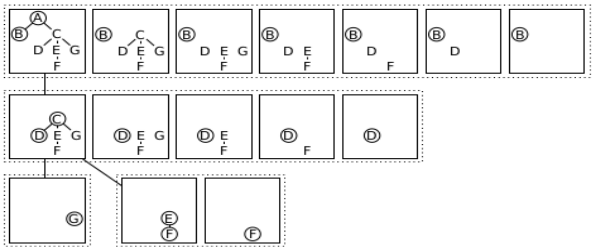
RTED: Robust Tree Edit Distance algoritmus

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

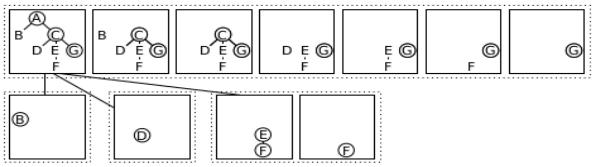
Funguje tak, že si predpočíta koľko podproblémov budeme musieť vyriešiť, ak použijeme stratégiu $left$, $right$, alebo $heavy$.

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

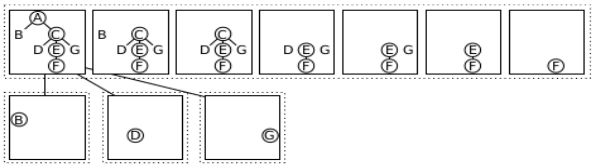
$$\mathcal{A}(\emptyset) = \emptyset$$
$$\mathcal{A}(F) = F \cup \mathcal{A}(F - r_L(F)) \cup \mathcal{A}(F - r_R(F))$$



(a) Left path decomposition (15 relevant subforests)



(b) Right path decomposition (11 relevant subforests)



(c) Heavy path decomposition (10 relevant subforests)

Obr. 2.3: Celková dekompozícia pomocou LRH strategii

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

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

Dôkaz. Demaine a kol. (2009) dokázali, že vzorec pre ťažké cesty je v poriadku. Rovnako tak, Zhang a Shasha (1989) to dokázali pre ľavé cesty. Jednoduchou úpravou vieme upraviť ich vzorec na použitie pravých ciest.

□

Definícia 9. *Minimálny počet podproblémov, ktoré potrebujeme vyrátať pri použití GTEDu je*

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

Dôkaz. je uvedený v Pawlik a Augsten (2011)

□

Namiesto $\mathcal{O}(n^3)$ rekurzie potrebujeme algoritmus, ktorý optimálnu stratégiu vyráta s nižšími časovými nárokmi ako potrebuje optimálny beh $GTED$u.

Popiseme teda algoritmus 3 - RTED, od tvorcov Pawlik a Augsten (2011). Bežiaci v čase $\mathcal{O}(n^2)$.

Prechádza vrcholmi v postorder, aby sa znížila pamäťová náročnosť algoritmu a nemuseli ukladať hodnoty medzi dvojicami relevant-subforest. Namiesto toho inkrementujeme hodnotu v rodičovskom vrchole pri každej návšteve jeho potomka.

Lemma 4. *Algoritmus 3 vyráta optimalnú LRH stratégiu pre dvojicu podstromov F a G a časová náročnosť algoritmu je $\mathcal{O}(n^2)$.*

Dôkaz. Toto tvrdenie dokázali Pawlik a Augsten (2011).

□

2.4 Mapovanie medzi stromami

Tabuľka vzdialenosti z $GTED$u medzi stromami F a G nám nebude stačiť. Potrebujeme vedieť ako strom F namapovať na G.

Princíp je v backtrackovani matice $ForestDistance$, teda zisťujeme, akú operáciu, sme v ktorom bode použili, podobne ako v zisťovaní operácií pri editačnej vzdialenosti reťazcov. Musíme ale používať $ForestDistance$ maticu, nie $TreeDistance$, keďže v nej sa odzrkadluje detailnejšia štruktúra stromov. Maticu $TreeDistance$ používame iba na počítanie single-path funkcie.

Algorithm 3 Optimálna stratégia

procedure RTED(F, G)
  $L_v, R_v, H_v \leftarrow$ polia velkosti $|F| \times |G|$
  $L_w, R_w, H_w \leftarrow$ polia velkosti $|G|$
  for all v postorder v F **do**
    for all w postorder v G **do**
      if v je list **then**
        $L_v[v, w] \leftarrow R_v[v, w] \leftarrow H_v[v, w] \leftarrow 0$
      if w je list **then**
        $L_w[w] \leftarrow R_w[w] \leftarrow H_w[w] \leftarrow 0$
      $C := \{$
        $(|F_v| \times \mathcal{A}(G_w) + H_v[v, w], \gamma^H(F)),$
        $(|G_w| \times \mathcal{A}(F_v) + H_w[w], \gamma^H(G)),$
        $(|F_v| \times \left|\mathcal{F}(G_w, \Gamma^L(G))\right| + L_v[v, w], \gamma^L(F)),$
        $(|G_w| \times \left|\mathcal{F}(F_v, \Gamma^L(F)\right|) + L_w[w], \gamma^L(G)),$
        $(|F_v| \times \left|\mathcal{F}(G_w, \Gamma^R(G))\right| + R_v[v, w], \gamma^R(F)),$
        $(|G_w| \times \left|\mathcal{F}(F_v, \Gamma^R(F))\right| + R_w[w], \gamma^R(G))$
        $\}$
      $(c_{min}, \gamma_{min}) \leftarrow (c, \gamma)$ take, ze $(c, \gamma) \in C \wedge c = min\{c' | (c', \gamma) \in C\}$
      $Strategies[v, w] := \gamma_{min}$
      if v nieje koren **then**
        UPDATE(L_v, v, w, c_{min}, $\gamma^L(parent(v))$
        UPDATE(R_v, v, w, c_{min}, $\gamma^R(parent(v))$
        UPDATE(H_v, v, w, c_{min}, $\gamma^H(parent(v))$
      if w nieje koren **then**
        UPDATE(L_w, w, c_{min}, $\gamma^L(parent(w))$
        UPDATE(R_w, w, c_{min}, $\gamma^R(parent(w))$
        UPDATE(H_w, w, c_{min}, $\gamma^H(parent(w))$
  return *Strategies*

procedure UPDATE($Table, v, w, c_{min}, \gamma$)
  $Table[parent(v), w] \stackrel{+}{=} \begin{cases} Table[v, w] & \text{ak } v \in \gamma \\ c_{min} & \text{v opacnom pripade} \end{cases}$

procedure UPDATE($Table, w, c_{min}, \gamma$)
  $Table[parent(w)] \stackrel{+}{=} \begin{cases} Table[w] & \text{ak } v \in \gamma \\ c_{min} & \text{v opacnom pripade} \end{cases}$

Algorithm 4 Počitanie mapovania

```
1: procedure MAPPING(F, G, TreeDistance)
2:     σ ← lubovolna LRH strategia
3:     ForestDistance ← SINGLE PATH(F, G, TreeDistance, σ)
4:     while F ≠ ∅ ∧ G ≠ ∅ do
5:         v ← UPDATE(F, σ)
6:         w ← UPDATE(G, σ)
7:         if ForestDistance[F, G] = ForestDistance[F − v, G] + C_del then
8:             Mapping ← Mapping ∪ (v → 0)
9:             F ← F − v
10:        else if ForestDistance[F, G] = ForestDistance[F, G−w]+C_ins then
11:            Mapping ← Mapping ∪ (0 → w)
12:            G ← G − w
13:        else
14:            if F a G su stromy then
15:                Mapping ← Mapping ∪ (v → w)
16:                F ← F − v
17:                G ← G − w
18:            else
19:                Mapping ← Mapping∪
20:                    MAPPING(F − F_v, G − G_w, TreeDistance)
21:                F ← F − F_v
22:                G ← G − G_w

23: procedure UPDATE(Forest, σ)
24:     γ ← cesta v lese Forest podla strategie σ
25:     return vrchol r_L(Forest) alebo r_R(Forest) alebo ∅ z Forest
26:         rovnako ako v definícií 7
```

3. Kreslenie molekuly

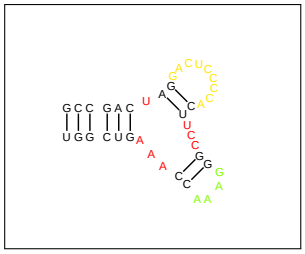
Po tom čo získame a aplikujeme mapovanie medzi šablonovou a cieľovou molekulou RNA, ziskame cieľovú molekulu s čiastočnou vizualizáciou, ktorej zvyšok treba dopočitať.

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

Sekundárna štruktúra RNA obsahuje množstvo motivov popisaných na obrázku 1.2. 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: cierne 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 pouzitie obrazka ako šablony, kedže 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 vkladaním alebo mazaním báz.

Jediné výnimky sú normalizácia vzdialensti medzi bázovými pármi a vyrovnávanie stemov.

3.2.1 Normalizácia vzdialeností v bázových pároch a vyrovnavanie stemov

Ako bolo uvedené, stemom rozumieme nevetviacu sa časť stromu tvorenú iba bázovými pármi.

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

Vyrovnávací algoritmus prechádza všetky stemy. Z ich začiatkov vedie priamku, na ktorej majú byť podľa pravidla uložené všetky stemove vrcholy. Rotáciami a posunutiami podstromov vieme docieliť 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.

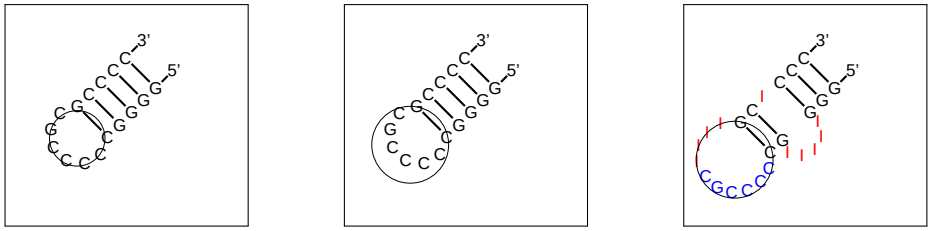
Algorithm 5 Rozloženie báz na kružnicu

```
1: procedure rozlozBazy(Begin, End, Bases)
2:     n ← veľkosť zoznamu báz Bases
3:     Γ ← dostatočne veľká kružnica pre n bodov prechádzajúca bodmi Begin
   a End
4:     Π ← rozdel kruhový obluk kružnice Γ od Begin po End na n bodov
5:     for all i in 1 .. n do
6:         nastav pozíciu bázy Bases[i] na bod Π[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 určenu pozíciu, t.j. nieje práve vložený then
4:             pripočítaj k pozicií 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: Priklad zvacsovania kruznice 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 bázové páry, alebo naopak ak sme niečo zmazali, tak dokáže celý podstrom pritiahnuť 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 jednodúche, potrebujeme iba použiť procedúru z algoritmu **??** s parametrami $Begin$ = požicia prvej bázy z bázového páru, End = požičia 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 posunuť celý podstrom aby nám dané bázy vošli. To vyriešime algoritmom 6. Následne nájdeme kružnicu a bázy na ňu naukladáme.

Vkladanie bázového 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 multibranch loop je zložitejšia ako všetky predchádzajúce prípady. Obrázky sú väčšinou ručné 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 roztiahnuť, alebo naopak priblížiť k sebe.

Ak sa jedná o pridanie/odobratie 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 rozdistribuovania vrcholov v loope, alebo posúvanie podstromu, ktoré sa deje v tomto prípade opačným smerom k predkovi.

4. TRAVeLer - Template RnA VisuaLization

Traveler je konzolova aplikacia programovana v C++ a je urceny pre operacne systemy UNIX-oveho typu. Vyvyjany a testovany bol na Linux-e a FreeBSD. Podpora ostatnych systemov nieje zarucena.

4.1 Instalacia

Pozadovane programove vybavenie je:

- gcc verzie aspon 4.9.2

Pri testovani boli zaznamenane problemy s regularnymi vyrazmi, ktore nam pomahaju pri nacitavani vstupnych suborov. Problem bol pri verzii gcc 4.7.2, ktora plne nepodporovala potrebne vyrazy.

Traveler prelozime zo zdrojovych kodov postupnostou prikazov z korenoveho adresara:

- cd src/
- make build
  Nasledne spustitelny subor je *src/build/traveler*.

4.2 Argumenty programu

Ak predpokladame, ze program lezi na $PATH$, spustame 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
```

Strucnu napovedu k programu dostaneme standardnym *-h* alebo *--help* argumentom.

Prepinacmi *--ted* a *--draw* vieme oddelit fazu pocitania vzdialenosti pomocou TEDu a nasledneho kreslenia.

Prepinac *--overlaps* po nakresleni obrazku v nom vyznaci vsetky miesta prekryvov, ak nejake vznikli. Zaroven ich pocet vypise do samostatneho suboru.

Nasledne rychlejsie dokazeme identifikovat molekuly, ktore potrebuju zvysenu pozornost.

Prepinac --*colored* aktivuje farebne zvyraznovanie zmien v strukture stromu oproti sablone. Pouzivame nasledovne kodovanie farbami:

- $Cervena$ vlozene bazy
- $Zelena$ editovane bazy
- $Modra$ bazy ktore sme potrebovali presunut
- $Hneda$ podstromy prekreslenych multibranch loop

Farbami zvyraznujeme zmeny v strome, to znamena, ze ak sa bazovy par zmeni v jednej baze, cely bude oznaceny ako editovany.

Modrou oznacujeme casti, ktore sme z nejakeho dovodu potrebovali presunut a prekreslit. Typickym prikladom je prekreslenie loopy po vlozeny/zmazani nejakej bazy. Vtedy sme vlozili napriklad 1 bazu ale potrebovali presunut dalsich 10 ktore uz v loop boli.

Hnedou farbou oznacujeme cele podstromy multibranch loopy, ktoru sme museli prekreslit. V tychto pripadoch vznikaju casto velke prekryvy a tymto ich odlisujeme od ostatnych, necakanych.

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 Rozšírenie podpory iných vstupných obrázkov.

4.2.1 Format fasta suboru

Ako format suborov kodujucich stromy pouzivame trochu upraveny fasta format.

Subor na prvom riadku obsahuje nazov molekuly hned za znakom > az po prvu medzeru. Na dalsich riadkoch obsahuje znaky sekvencie RNA a znaky kodujuce sekundarnu strukturu. Je zvykom, ze riadky su siroke najviac 80 znakov.

Fasta subor pre sablonovu molekulu RNA potrebuje iba nazov a zatvorkovanie, pre vizualizovanu molekulu aj sekvenciu. Je to dane tym, ze sekvenciu si vieme vybrat z obrazka sablony.

4.3 Priklad vstupu

Teraz uvedieme priklad vstupu pre malu podjednotku ribozomalnej RNA mysi, konkretne priklad fasta suboru 4.1, podporovaneho formatu post script suboru 4.2 a nasledne aj obrazok vizualizacie v post script subore 4.3.

Poznámka. Podporujeme iba jeden format PostScript súborov - ten používa databáza CRW publikovaná Cannone a kol. (2002). Ďalšie rozšírenia podpory inych formatov rozoberáme v kapitole Rozšírenie podpory iných vstupných obrázkov.

```
>mouse
UACCUGGUUGAUCCUGCCAGUAGCAUAUGCUUGUCUCAAAGAUUAAGCCAUGCAUGUCUAAGUACGCACGGCCGGUACAG
UGAAACUGCGAAUGGCUCAUUAAAUCAGUUAUGGUUCCUUUGGUCGCUCGCUCCUCUCCUACUUGGAUAACUGUGGUAAU
UCUAGAGCUAAUACAUGCCGACGGGCGCUGACCCCCCUUCCCGGGGGGGGAUGCGUGCAUUUAUCAGAUCAAAACCAACC
CGGUGAGCUCCCUCCCGGCUCCGGCCGGGGGUCGGGCGCCGGCGGCUUGGUGACUCUAGAUAACCUCGGGCCGAUCGCAC
GCCCCCCGUGGCGGCGACGACCCAUUCGAACGUCUGCCCUAUCAACUUUCGAUGGUAGUCGCCGUGCCUACCAUGGUGAC
CACGGGUGACGGGGAAUCAGGGUUCGAUUCCGGAGAGGGAGCCUGAGAAACGGCUACCACAUCCAAGGAAGGCAGCAGGC
GCGCAAAUUACCCACUCCCGACCCGGGGAGGUAGUGACGAAAAAUAACAAUACAGGACUCUUUCGAGGCCCUGUAAUUGG
AAUGAGUCCACUUUAAAUCCUUUAACGAGGAUCCAUUGGAGGGCAAGUCUGGUGCCAGCAGCCGCGGUAAUUCCAGCUCC
AAUAGCGUAUAUUAAAGUUGCUGCAGUUAAAAAGCUCGUAGUUGGAUCUUGGGAGCGGGCGGGCGGUCCGCCGCGAGGCG
AGUCACCGCCCGUCCCCGCCCCUUGCCUCUCGGCGCCCCCUCGAUGCUCUUAGCUGAGUGUCCCGCGGGGCCCGAAGCGU
UUACUUUGAAAAAAUUAGAGUGUUCAAAGCAGGCCCGAGCCGCCUGGAUACCGCAGCUAGGAAUAAUGGAAUAGGACCGC
GGUUCUAUUUUGUUGGUUUUCGGAACUGAGGCCAUGAUUAAGAGGGACGGCCGGGGGCAUUCGUAUUGCGCCGCUAGAGG
UGAAAUUCUUGGACCGGCGCAAGACGGACCAGAGCGAAAGCAUUUGCCAAGAAUGUUUUCAUUAAUCAAGAACGAAAGUC
GGAGGUUCGAAGACGAUCAGAUACCGUCGUAGUUCCGACCAUAAACGAUGCCGACUGGCGAUGCGGCGGCGUUAUUCCCA
UGACCCGCCGGGCAGCUUCCGGGAAACCAAAGUCUUUGGGUUCCGGGGGGAGUAUGGUUGCAAAGCUGAAACUUAAAGGA
AUUGACGGAAGGGCACCACCAGGAGUGGGCCUGCGGCUUAAUUUGACUCAACACGGGAAACCUCACCCGGCCCGGACACG
GACAGGAUUGACAGAUUGAUAGCUCUUUCUCGAUUCCGUGGGUGGUGGUGCAUGGCCGUUCUUAGUUGGUGGAGCGAUUU
GUCUGGUUAAUUCCGAUAACGAACGAGACUCUGGCAUGCUAACUAGUUACGCGACCCCCGAGCGGUCGGCGUCCCCCAAC
UUCUUAGAGGGACAAGUGGCGUUCAGCCACCCGAGAUUGAGCAAUAACAGGUCUGUGAUGCCCUUAGAUGUCCGGGGCUG
CACGCGCGCUACACUGACUGGCUCAGCGUGUGCCUACCCUGCGCCGGCAGGCGCGGGUAACCCGUUGAACCCCAUUCGUG
AUGGGGAUCGGGGAUUGCAAUUAUUCCCCAUGAACGAGGAAUUCCCAGUAAGUGCGGGUCAUAAGCUUGCGUUGAUUAAG
UCCCUGCCCUUUGUACACACCGCCCGUCGCUACUACCGAUUGGAUGGUUUAGUGAGGCCCUCGGAUCGGCCCCGCCGGGG
UCGGCCCACGGCCCUGGCGGAGCGCUGAGAAGACGGUCGAACUUGACUAUCUAGAGGAAGUAAAAGUCGUAACAAGGUUU
CCGUAGGUGAACCUGCGGAAGGAUCAUUA
...(((((.......))))).((((((((((.(((((((((.....(((.(((..((...(((....((..........)
)...)))))......(((......((((..((..((....(((.................((((....(((((((....
.)))))))).....))))........((((...((((((......))))))...))))....(((((.......(((((.(
(((...((((.(((((((((....))))))))..))))).))))....)))))......))))))...........((((.(
(((......))))))))....)))...))))..))))(((..(.(((....(((((((.......)))))))))))...
..))))...(((((((((....))))...)))))))..((((((..........))))))..((((....))))...))))))
.).....(.(((...(((((...)))))..))))..)).))))))....(((((((((.(((....)))..)))))).)))..
.....(((.(((.......))))..))..........((((((.......((((.....((....))).......))))))))
)).)))))))))..........(((.......((((...(((.......(((.(((((((((((((.((((....))))
....)))))))))..)))))))).......((((.(((((...(((((((......)))))))....)))))))))).....
................................................................................
.........................................((((((((..(((((((((..(((((((...(((.
.....)))......))))))))..))....(..((....))))))))))..)))))..)))...))))....(((
(((...((...((((.........))))...))))))))..........((((((.(((..(((((((((.(((((....)
))))))))))))..)))...((....))...)))....)))..)))..(((.....((((....))))....))).......
..(((((.(((((((..((..(((((((((((((((((....((((........))))........(((((((....(((((
(........((((((........))))))......)))))...((.((((..(((((((((...(((((((((....)))
..((((......))))..))))))).....((((.(((.((((..((((....(((..((((....)))).)))....)))
)..)))))))..(((((((.....))))))))....))))...)))).)))...).)))))))).....)))))))...)
).)))))).))...(((((((.....(((.......((..((((....))))..))).....)))).....)))))))...
...(...(((((((((........)))))))...).....)))))).....(((((((.......)))))))).....)
)...))))))))))..))....((.((.(.(((((((((.((.(((((((((((((..(((((((((((((((.(((((((((
(((.....))))))))))))...)))))))))))))))..)))))))))))))).)))))))))..).))..))....(((
(((((((....)))))))))).........
```

Obr. 4.1: Priklad fasta suboru

```
%!
/lwline {newpath moveto lineto stroke} def
  ...
434.00 -129.00 422.00 -138.00 lwline
0.00 setlinewidth
446.00 -421.00 446.00 -412.00 lwline
306.00 -283.00 306.00 -273.00 lwline
  ...
(U) 303.30 -273.00 lwstring
(A) 303.30 -265.00 lwstring
(C) 303.30 -257.00 lwstring
(C) 303.50 -248.68 lwstring
(U) 311.24 -246.68 lwstring
(G) 318.99 -244.68 lwstring
  ...
```

Obr. 4.2: Priklad podporovaneho formatu post script suboru

4.4 Vystupne subory

Program generuje 2 druhy vystupov. Prvym je ulozenie tabulky mapovania TED algoritmu a druhym su obrazky vo formate SVG a PS.

Oznacme $T1$ strom sablony a $T2$ vizualizovany strom.

Format mapovacieho suboru je nasledovny:
Prvy riadok obsahuje $DISTANCE : n$, kde n je editacna vzdialenost medzi $T1$ a $T2$.

Ostatne riadky su vo formate $i\ j$, kde $i, j \geq 0$. Inymi slovami:

- 1 2 - prvy vrchol z $T1$ potrebujeme namapovat na druhy vrchol z $T2$
- 0 2 - do vysledneho stromu vkladame druhy vrchol z $T2$
- 1 0 - zo stromu $T1$ mazeme prvy vrchol

PostScript subor je zlozeny z hlavicky v ktorej su definicie kresliacich funkcii za ktorymi su riadky kreslenia molekuly. Priklad je na obrazku 4.4.

Najprv definujeme operacie kreslenia v hlavicke suboru - *lwline*, *lwstring* a *lwarc* - kreslenie ciar, textu a kruznic. Za ktorymi nasleduje samotne kreslenie molekuly.

Podobne funguje kreslenie v SVG subore ktoreho priklad je na obrazku 4.5. Elementy $< text >$ vypisuju na danu poziciu text, $< line >$ naopak kreslia ciary a $< circle >$ zase kruznice.

4.5 Rozšírenie podpory iných vstupných obrázkov

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

Definícia 10. *Extractor bude nejaky objekt ktory vie zo suboru urciteho typu vynat potrebne polozky reprezentujuce RNA sekvenciu a poziciu baz na obrazku.*

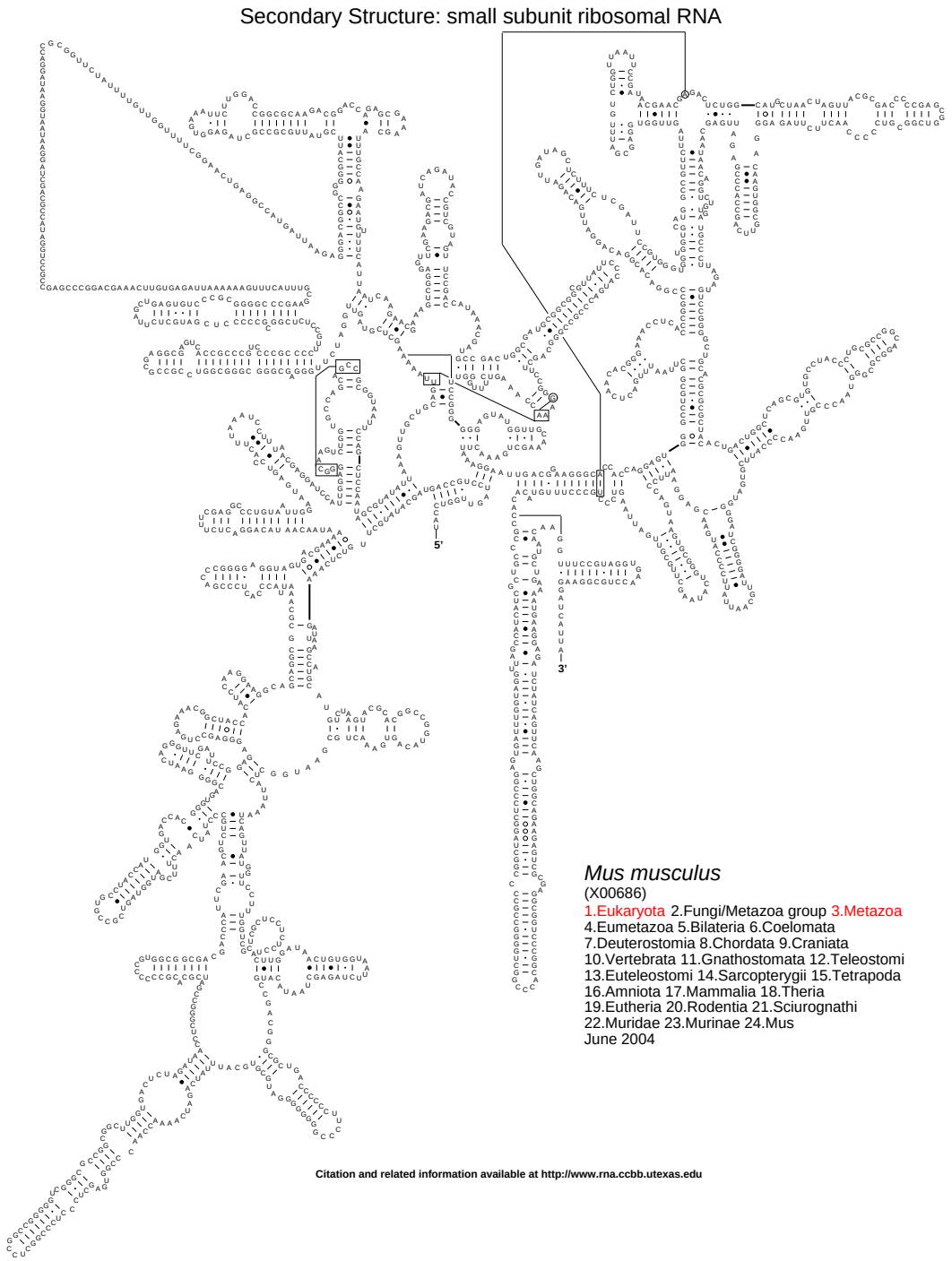
Pri tvorbe aplikácie sme už mysleli na budúcnosť a načítavanie súboru robíme v jednom lahko rozsiritelnom module. Ten sa na zaklade typu v parametre $--template-tree$ rozhoduje aky extractor pouzit. Predvoleny a jediny implementovany je PostScript extractor fungujuci nad subormi z CRW databazy.

Na implementovanie extractora potrebujeme implementovat existujuce rozhranie *extractor*, co znamena implementovat metodu *init* s parametrom nazvu suboru.

Jej uloha je zo suboru ziskat sekvenciu RNA a pozicie baz.

Poslednou ulohou je pridat dvojicu $(nzov\_exctractora, extractor)$ do tabulky implementovanych v metode $create\_exctractors()$.

Naslednym volanim $--template-tree$ $--type\ nazov\_extractora$ zacneme pouzivat nas novo implementovany extractor.



Obr. 4.3: Priklad vstupneho obrazka

```
%!
/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: Format vystupneho SVG suboru

5. Vysledky prace

V tejto kapitole zhrnieme vysledky ktore nasa praca dosiahla.

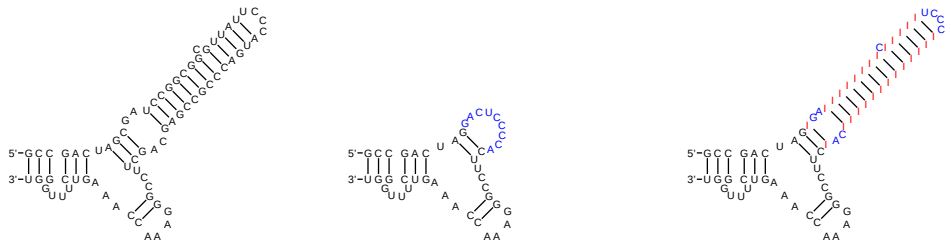
Uz v predchadzajucich kappitolach sme sa stretli s niekolkymi prikladmi, napriklad ako si program poradil s vkladanim do hairpinu - obrazok 3.2.

Na dalsom (obrazok 5.1) simulujeme mazanie s naslednym vkladanim, teda 2 k sebe inverzne operacie. Po zmazaní bazovych parov na hornej vetve molekuly, sa nam vsetky neparove bazy zliali a vytvorili jednu loop. Nasledne po opatovnom vlozeni bazovych parov (pre lepsie zviditelnenie sme ich oznacili Ï"), vznikla struktura velmi podobna predchadzajucej.

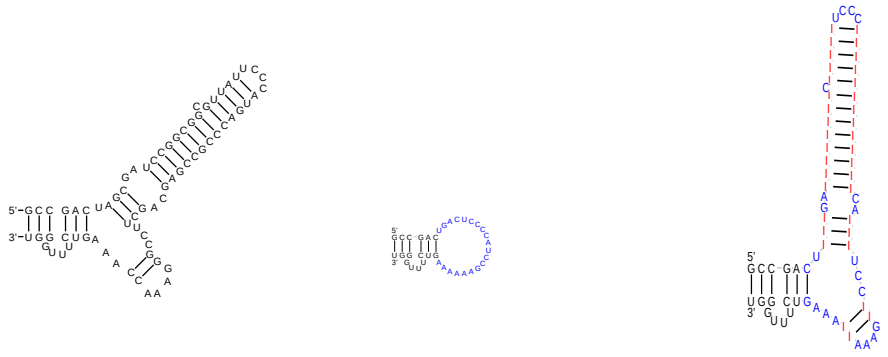
Obrazok ma za ciel ukazat, ze vieme znovu nakreslit povodnu strukturu iba s malymi zmenamy v pozicii nukleotidov (vysledne loopy su trochu plytsie ako povodne).

Rovnako obrazok 5.2 rekonstruuje vetvenie sa stromu. Ako je vidiet, v tomto obrazku je uz viac rozdielov, vychylenie je celkom badatelne.

Na takto malych castiach bez velkych vetveni nam ani taketo zmeny nevadia. Pri velkych molekulach ako ukazeme neskor problemy nastavaju.



Obr. 5.1: Inverzne operacie: rekonstrukcia stemu

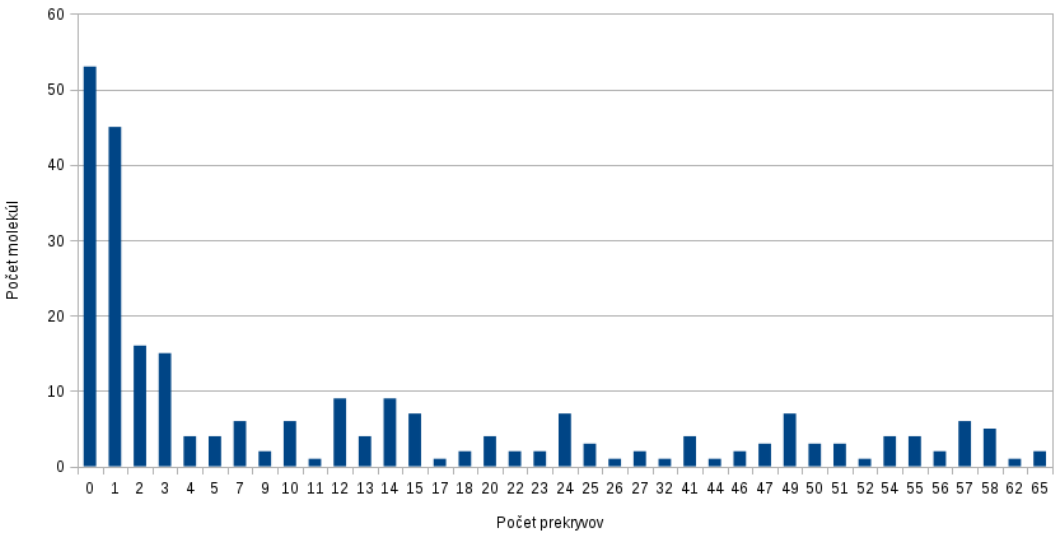


Obr. 5.2: Inverzne operacie: rekonstrukcia multibranch loop

Ako dalsie, testovali sme schopnost nasho algoritmu vizualizovat znamu podjednotku 16S ribozomalnej RNA na zivocisnej risi. CRW databaza obsahuje 16 organizmov so znamou sekundarnou strukturou.

Ribozomalna RNA bola vybrata lebo je v centre zaujmu mnohych vyskumov a taktiez kvoli jej velkosti a zlozitosti.

Nas vizualizacny test sme spustili na vsetky pary RNA, z ktorych sme ziskali 256 vizualizacii.



Obr. 5.3: Počet prekryvov v testovaných molekulách

Na obrazku 5.3 vidime pocty molekul s danym poctom prekryvov. Je ale niekolko typov molekul u ktorych sme nejake prekryvy cakali - napriklad, ak sme v nej potrebovali prekreslit multibranch loop. Takuto zavislost nam vyjadruju dalsie dva grafy, prvy - 5.5 nam ukazuje, ze ak program musel rotovat a prekreslovat multibranch loopy, nedarilo sa mu najlepsie. Naopak, ak z prveho grafu odoberieme molekuly, ktore museli pouzit rotacie - graf 5.4, vidime, ze algoritmus sablonovej vizualizacie si viedol celkom dobre, prekryvy vznikali iba ojedinele.

Z tabulky 5.1 je vidiet, ze pocet prekryvov zavisi od poctu operacii vkladania a mazania ktore v molekule musime urobit. Statistika pracuje s prvou, piatou, desiatou a patnastou najblizsou molekulou z pohladu $tree - edit - distance$.

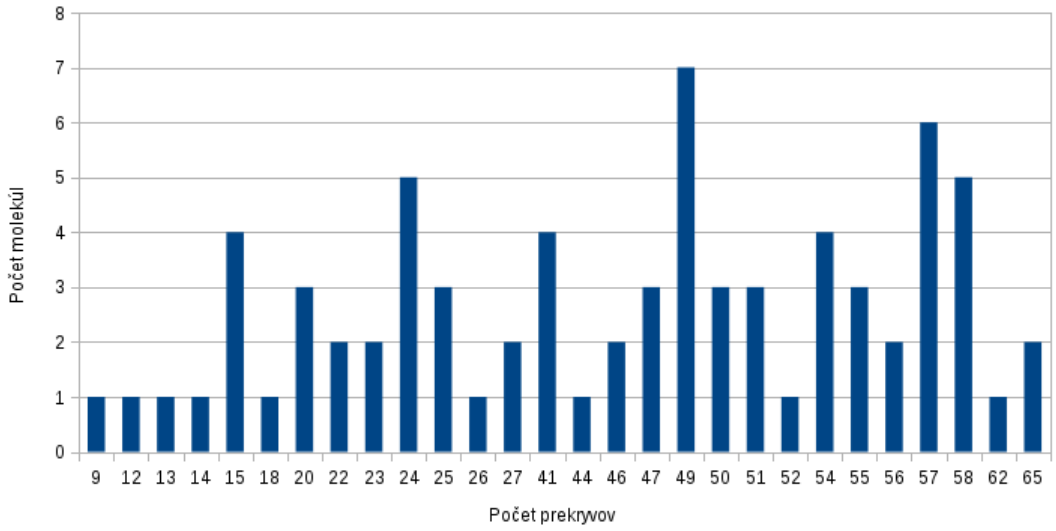
Zaujimavostou je, ze ako najvzdialenejsiu molekulu (v poradi patnastu) si vsetci vybrali molekulu od jedneho konkretneho zastupcu *echinococcus_granulosus* a vzdialenost je 805,63 s odchylkou 12,92.

5.1 Celkove vysledky

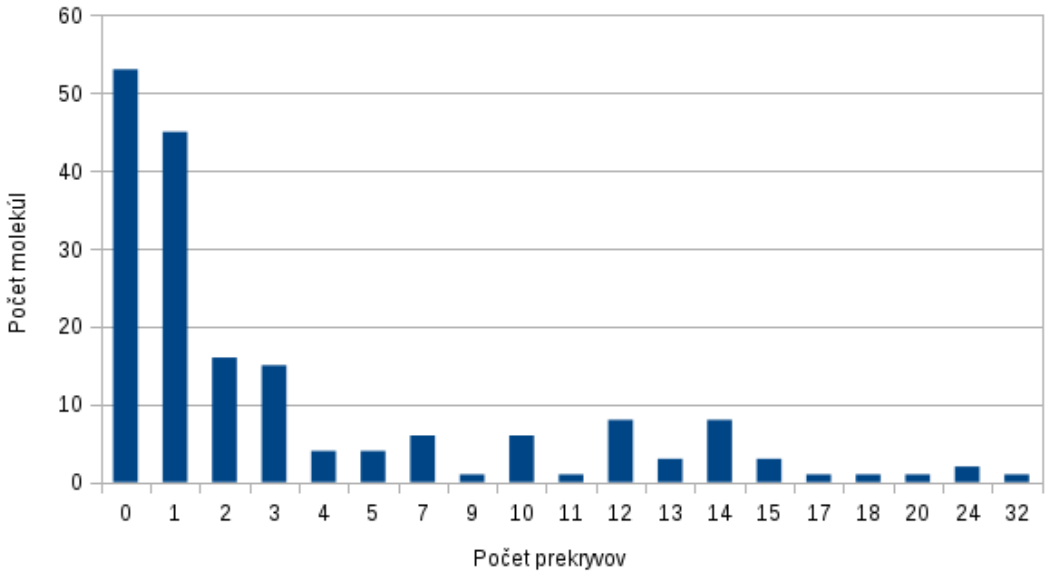
V tejto kapitole uvedieme vygenerovane obrazky niektorych molekul a na nich ukazeme caste problemy, ktore pri vizualizacii nastavali.

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

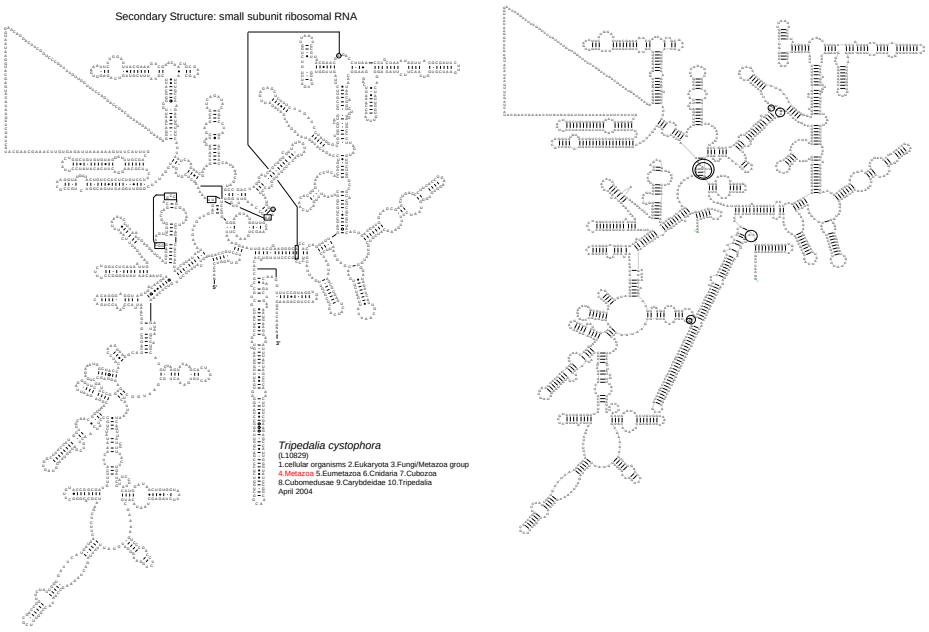
Tabuľka 5.1: Počty prekryvov v závislosti od tree-edit-distance vzdialenosti



Obr. 5.4: Počet prekryvov: molekuly ktore potrebovali prekreslit multibranch loop



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



Obr. 5.6: Chyba pri otoceni vetvy

5.1.1 Otacanie vetvy kvoli existujucej hrane

Jednym prikladom za vsetky je molekula zivocicha *Tripedaliacystophora* - meduzy. Po tom, co sme dali molekulu nakreslit samu na seba vznikol problem, ze cela jedna vetva molekuly sa otocila na jednu stranu. Je to sposobene existenciou bazoveho paru, ktory je v povodnej molekule znazorneny dlhsou lomenou ciarou.

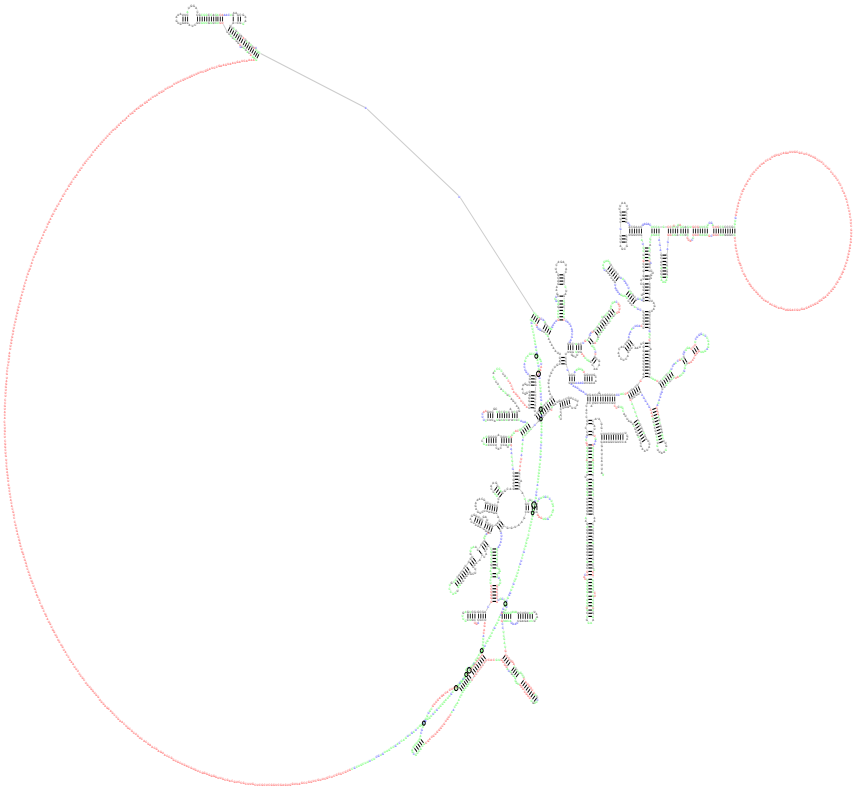
Kedze nas program vsetky vzdialenosti normalizuje a nasledne uklada bazy stemu na jednu priamku, vznikaju obrazky podobne 5.6.

5.1.2 Rozlozenie baz na kruznicu

Niekedy sa prekresleniu celej loop nevyhneme. Ak napriklad vkladame velmi velke mnozstvo baz na jedno miesto, dochadza k problemom nacrtnutym na obrazku 5.7.

Na tomto konkretnom priklade je nakresleny stem, vo vnutri ktoreho je velka loop. Kvoli tomu, ze chceme dodrziavat pravidla o kruznicovom tvare loopy, najdeme kruznicu dostatocne velku. V tomto pripade az priliz velku.

Poznamka - vo vrchnej vetve ja taktiez znazornena kruznica, ale na rozdiel od spodnej obsahuje iba 2 vrcholy.



Obr. 5.7: Chyba pri rozkladani baz na kruznicu

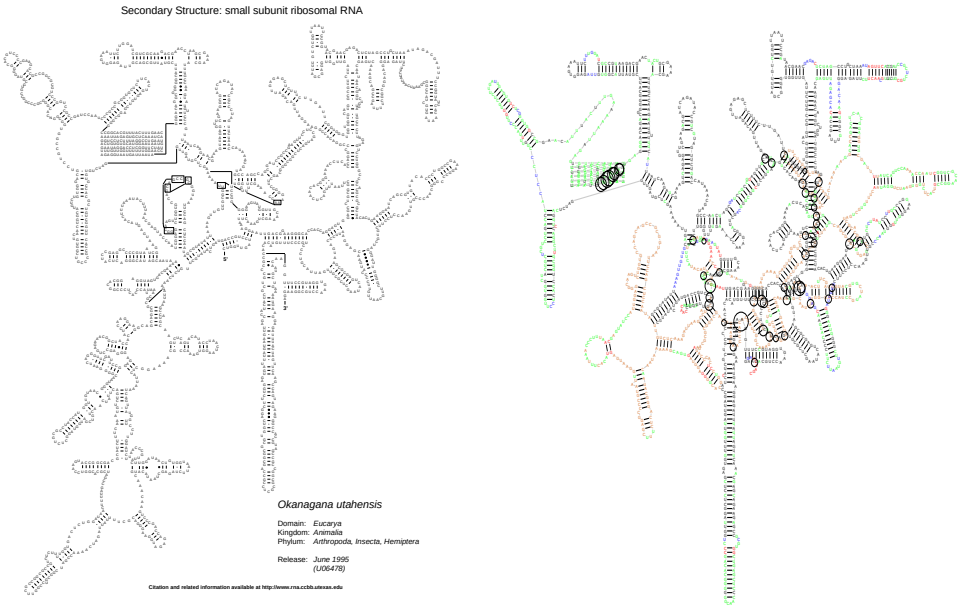
5.1.3 Otacanie vetvy kvoli prekreslovaniu multibranch loopy

Ako uz aj graf na obrazku 5.4 ukazal, prekreslovanie multibranch loopy a rotacie vsetkych vetiev sposobuje masivne prekryvy.

Pripajame jeden priklad na obrazku 5.8. Miesto vlavo dolu, kde zacinaju bazy sa sfarbovat na hnedo, je multibranch loop, ktoru sme potrebovali z dovodu vlozenej novej vetvy (oznacena cerveno) prekreslit.

Vysledok je taky, ze vsetky vetvy sme ulozili na kruznicu a pootacali do vhodneho smeru a tym vzniklo vela prekryvov.

Na obrazku je vidiet este jednu vec, oznacene krizenia vo vyslednom obrazku vlavo hore. V kapitole o upravach multibranch loop sme spominali, ze prekresleniu celej loopy sa snazime vyhnut ak to ide. Predpokladame, ze ak je baz vela a zmeny male, bazy trochu poposuvame aby sa novy vrchol zmestil medzi ne, alebo prave naopak ich roztiahneme, aby sme vyplnili medzeru po starom. Kvoli tomu vyzera tato struktura tak pomiesane a kvoli tomu na tomto mieste vznikaju dalsie prekryvy.



Obr. 5.8: Chyba pri otacani kvoli 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.

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.

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.

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.

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.

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.

Zoznam obrázkov

Zoznam tabuliek

Seznam použitých zkratek

Přílohy