

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 štruktúry RNA a grafov	4
1.1 Co je RNA	4
1.2 Sekundarna štruktúra rRNA + konzervovanosť	4
1.2.1 Motívy	6
1.3 Grafové pojmy	6
1.3.1 Značenie	6
1.4 Stromová reprezentácia sekundárnej štruktúry	7
2 Tree-edit-distance algoritmus	8
2.1 Hlavná myšlienka TED-u	8
2.2 Značenie	8
2.3 Algoritmy dynamického programovania	9
2.3.1 RTED	9
2.4 Mapovanie medzi stromami	14
3 Kreslenie molekuly	17
3.1 Štruktúry v RNA	17
3.2 Algoritmus	18
3.2.1 Normalizácia vzdialeností v báзовých pároch a vyrovna- vanie stromov	18
3.2.2 Operácie na stromoch	18
3.2.3 Vkladanie nového vrcholu do stromu	19
3.2.4 Modifikácia multibrach loop	19
3.2.5 Mazanie vrcholu zo stromu	20
4 TRAVeLer - Template RnA Visualization	21
4.1 Inštalácia	21
4.2 Argumenty programu	21
4.2.1 Formát fasta súboru	22
4.3 Príklad vstupu	22
4.4 Výstupné súbory	24
4.5 Rozšírenie podpory iných vstupných obrázkov	24
5 Výsledky práce	27
5.1 Celkové výsledky	28
5.1.1 Otáčanie vetvy kvôli existujúcej hrane	30
5.1.2 Rozloženie báz na kružnicu	30
5.1.3 Otáčanie vetvy kvôli prekreslovaniu multibrach loopy	31
Záver	33
Seznam použité literatury	34
Zoznam obrázkov	35

Zoznam tabuliek	36
Seznam použitých zkratk	37
Přílohy	38

Úvod

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

1. Uvod do štúdia štruktúry RNA a grafov

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

1.1 Co je RNA

Ribonukleónová kyselina je nukleónová kyselina tvorená jedným vláknom kovalentne naviazaných ribonukleotidov, ktoré sú základnými stavebnými jednotkami nukleónových 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 adenín (A), guanín (G), cytozín (C), uracyl (U), pri DNA sa namiesto uracylu vyskytuje tymín (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.

Štruktúru nukleových kyselín môžeme chápať 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 dvojvláknová molekula, u ktorej spojenie medzi vláknami sa realizuje na princípe komplementarity. Naopak, RNA je iba jednovláknová molekula a v snahe minimalizovať voľnú energiu molekuly, sa paruje sama na seba. V tomto hrajú rolu prítahové sily medzi bázami.

V práci budeme štruktúrou myslieť práve sekundárnu štruktúru 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 aminokyselín v ribozome bunky (tRNA). Avšak existuje mnoho ďalších, od relatívne malých molekúl tvorených desiatkami báz, ktoré pomáhajú pri expresii genov (miRNA, siRNA, tmRNA a ďalšie), až po veľké, tvorené tisíckami nukleotidov (rRNA).

1.2 Sekundárna štruktúra rRNA + konzervovanosť

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

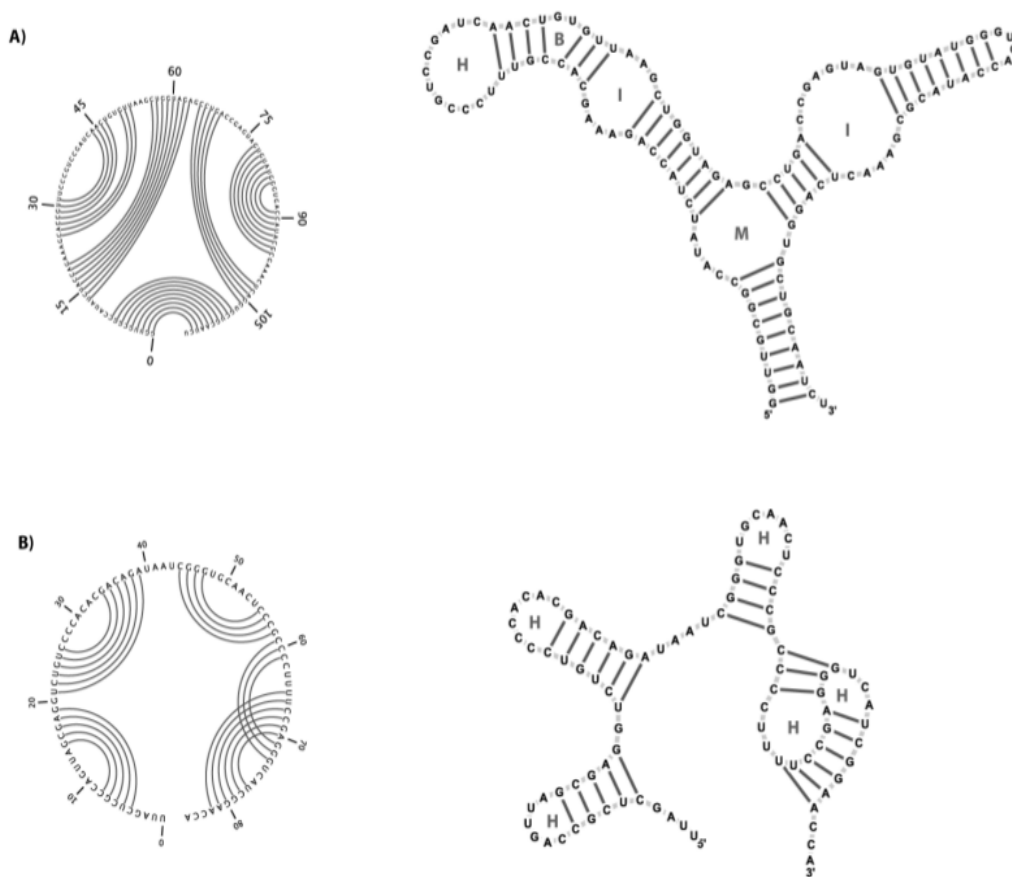
Definícia 1 (Primárna štruktúra RNA). *Nech Σ je abeceda $\{A, C, G, U\}$. Potom slovo $W \in \Sigma^n$ nad touto abecedou je sekvencia nukleotidov (baz) RNA.*

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

Definícia 2 (Sekundárna štruktúra RNA). *Nech W je sekvencia podľa definície 1 dĺžky n . Sekundárnou štruktúrou označíme množinu \mathbb{S} parov nukleotidov (i, j) takých, že pre dva pary (i, j) a $(k, l) \in \mathbb{S}$ (bez ujmy na obecnosti $i \leq k$) platí jedno z nasledujúcich:*

- $i = k \iff j = l$
- $i < j < k < l$, cize par (i, j) predchádza 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 usporiadaní párov, buď sú na sebe nezávislé alebo na seba nadväzujú. Posledná podmienka zakazuje existenciu pseudouzlov (pseudoknots). Pseudouzol patrí medzi najčastejšie typy priestorového usporiadania RNA. Vytvára niekoľko interakcií vrámci jednej molekuly a smyčky typu loops, ktoré vznikajú medzi rôznymi molekulami.

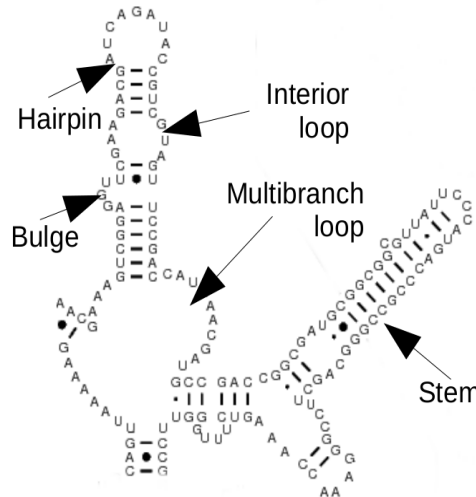


Obr. 1.1: Circular Feynman - kruhová reprezentácia sekundárnej štruktúry

1.2.1 Motivy

Motivom v RNA máme na mysli časti molekuly, ktoré vytvárajú určité štruktúry. Na obrázku 1.2 vidíme motivy, ktoré sa môžu v RNA vyskytovať.

Stem (stonka) je časť molekuly kde sa na seba parujú dva súvislé časti RNA vlákna. Interior loop spája dva stemy a medzi nimi na oboch stranách obsahuje nespárované bázy. Podobná je bulge (vypuklina), ale nespárované nukleotidy má 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 rozprávaní ámam bude stačiť rozdelenie na stem a loop.



Obr. 1.2: Strukturalne motivy v RNA

1.3 Grafové pojmy

Potrebuje 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. *Usporiadáný 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. Navyše 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 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 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 reprezentácia sekundarnej štruktúry

Definícia 2 nám ponúka reprezentovať sekundárnu štruktúru ako usporiadaný strom.

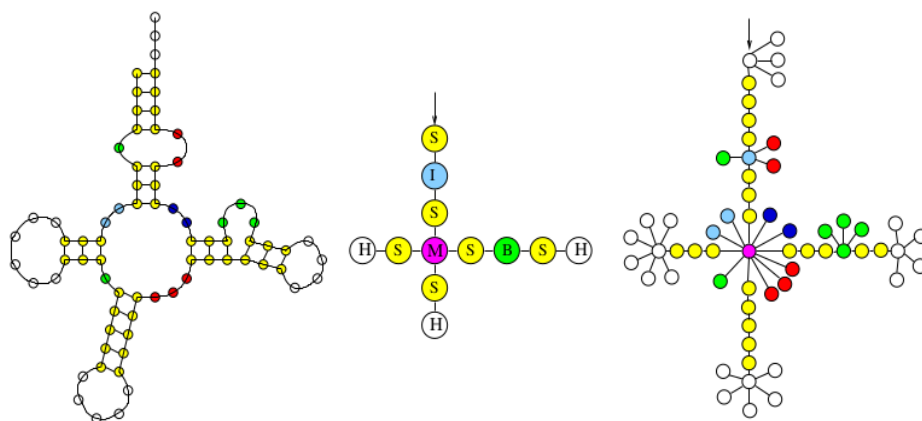


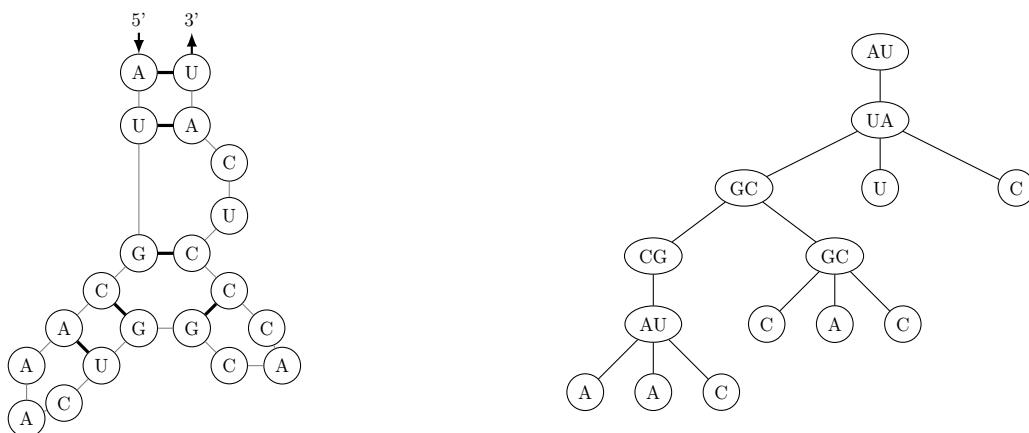
Figure 3: A secondary structure and its tree representations.

Obr. 1.3: Varianty reprezentácie 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 bázy pár. Príklady možno vidieť na obrázku 1.3.

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

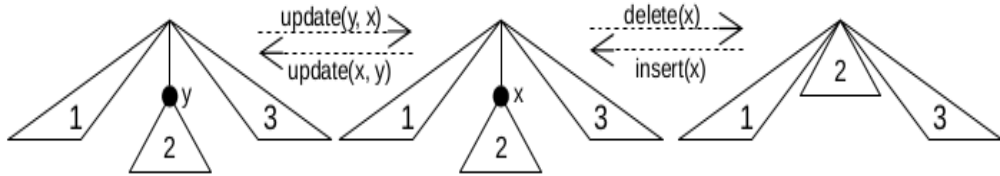


2. Tree-edit-distance algoritmus

Jadro aplikácie 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 Levenstheinovoho string-edit-distance algoritmu. Problém u reťazcov je špeciálnym 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 definovaná ako minimálny počet editačných operácií, ktoré z F urobia G . Používame štandardne editačné operácie - delete, insert, update.



Obr. 2.1: Ukazky TED operácii

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

2.2 Značenie

V tejto kapitole sa budeme riadiť značením Pawlik a Augsten (2011). Teda, používame definíciu 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 \bar{F} s vrcholmi $N_{\bar{F}} \subseteq N_F$ a hranami $E_{\bar{F}} \subseteq E_F \cap N_{\bar{F}} \times N_{\bar{F}}$. 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. Editačná 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).

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

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

2.3 Algoritmy dynamického programovania

Tai (1979) predstavil algoritmus s priestorovou a časovou zložitou $\mathcal{O}(m^3 \cdot n^3)$, Zhang a Shasha (1989) algoritmus následne vylepšili pozorovaním toho, že nepotrebuje vzdialenosti medzi všetkými pármami podlesov. Algoritmus mal 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 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 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. Zovšeobecniili 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 definícia 6) a GTED (General Tree Edit Distance) algoritmus samotný výpočet rekursie 2.2 s aplikovaním danej stratégie.

Definícia 6 (Dekompozičná stratégia). *Nech F a G sú lesy. Dekompozičná stratégia v rekursii 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 podstromi 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:    $\sigma \leftarrow S[F, G]$ 
3:   if  $\sigma \in \sigma^*(F)$  then
4:     for all  $F' \in F - \sigma$  do
5:        $TreeDistance \leftarrow TreeDistance \cup \text{GTED}(F', G, TreeDistance, S)$ 
6:      $TreeDistance \leftarrow TreeDistance \cup$ 
7:        $\text{COMPUTE\_DISTANCE}(F, G, TreeDistance, \sigma)$ 
8:   else
9:      $TreeDistance \leftarrow TreeDistance \cup (\text{GTED}(G, F, TreeDistance^T, S^T))^T$ 
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 dekompozícnou cestou a stromom G .

Následne pre všetky relevant-subtrees (viz definície 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*

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

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 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 doslednému usporiadaniu lesov si v každom kroku pripravíme potrebné data pre ďalší krok algoritmu 2.

Algorithm 2 Single path function

```
1: procedure COMPUTE DISTANCE( $F, G, TreeDistance, \sigma$ )
2:   if  $\sigma \in \sigma^*(F)$  then
3:     for all  $G' \in \text{RELEVANT SUBTREES}(G)$  do
4:       SINGLE PATH( $F, G', TreeDistance, \sigma$ )
5:   else
6:     for all  $F' \in \text{RELEVANT SUBTREES}(F)$  do
7:       SINGLE PATH( $F, G, TreeDistance, \sigma$ )
8:
9: procedure SINGLE PATH( $F, G, TreeDistance, \sigma$ )
10:   $ForestDistance \leftarrow$  empty array  $|F| + 1 \times |G| + 1$ 
11:   $ForestDistance[\emptyset][\emptyset] := 0$ 
12:  for  $F'$  subforest in GET ORDERED SUBFORESTS( $F, \sigma$ ) do
13:     $Last_F \leftarrow$  last added node to  $F'$ 
14:     $ForestDistance[F'][\emptyset] := ForestDistance[F' - Last_F][\emptyset] +$ 
15:       $C_{del}(Last_F)$ 
16:  for  $G'$  subforest in GET ORDERED SUBFORESTS( $G, \sigma$ ) do
17:     $Last_G \leftarrow$  last added node to  $G'$ 
18:     $ForestDistance[\emptyset][G'] := ForestDistance[\emptyset][G' - Last_G] +$ 
19:       $C_{ins}(Last_G)$ 
20:  for  $F'$  subforest in GET ORDERED SUBFORESTS( $F, \sigma$ ) do
21:    for  $G'$  subforest in GET ORDERED SUBFORESTS( $G, \sigma$ ) do
22:       $Last_F \leftarrow$  last added node to  $F'$ 
23:       $Last_G \leftarrow$  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, keď aspon jeden z lesov nieje stromom, tak cenu medzi F_{Last_F} a G_{Last_G} máme vyrátanú z predchádzajúcich krokov, alebo z inej vetvy rekurzcie.

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 dekompozícii 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, takže nikdy sa nedostanem do menšieho aby som mohol mu znova nastaviť hodnotu. To iste platí aj pre *ForestDistance*.

□

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

Dôkaz. Hodnota *ForestDistance* pre použitie s prázdny m 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 dekompozícii 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.

□

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

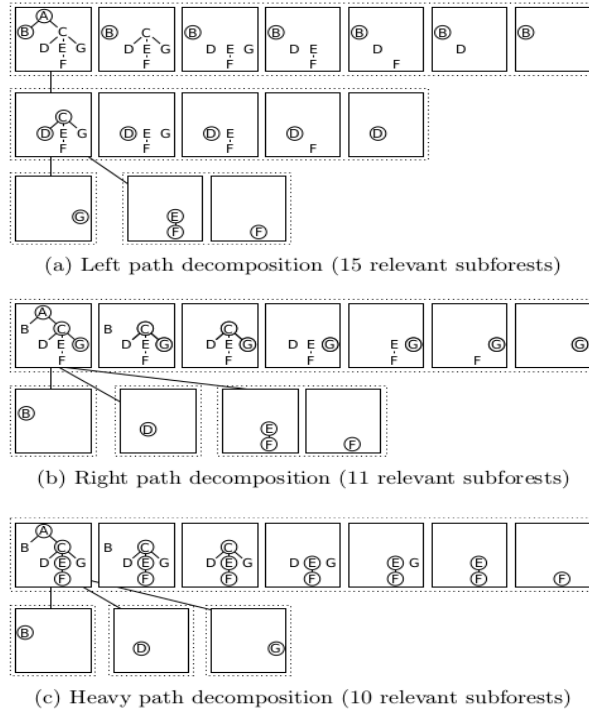
RTED budeme vnímať ako algoritmus na výpočítanie optimálnej stratégie - teda algoritmus, ktorý nám poradí ako najlepšie dekomponovať obidva stromy.

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 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.

$$\mathcal{A}(\emptyset) = \emptyset$$

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



Obr. 2.3: Celková dekompozícia pomocou LRH stratégiei

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

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

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*

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

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 GTEDu.

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 optimálnu 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 GTEDu medzi stromami F a G nám nebude stačiť. Potrebujeme vedieť ako strom F namapovať na G .

Princíp je v backtrackovaní 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 odzrkadľuje detailnejšia štruktúra stromov. Maticu *TreeDistance* používame iba na počítanie single-path funkcie.

Algorithm 3 Optimálna stratégia

```

1: procedure RTED( $F, G$ )
2:    $L_v, R_v, H_v \leftarrow$  polia veľkosti  $|F| \times |G|$ 
3:    $L_w, R_w, H_w \leftarrow$  polia veľkosti  $|G|$ 
4:   for all  $v$  postorder v  $F$  do
5:     for all  $w$  postorder v  $G$  do
6:       if  $v$  je list then
7:          $L_v[v, w] \leftarrow R_v[v, w] \leftarrow H_v[v, w] \leftarrow 0$ 
8:       if  $w$  je list 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:       $(c_{min}, \gamma_{min}) \leftarrow (c, \gamma)$  take, ze  $(c, \gamma) \in C \wedge c = \min\{c' | (c', \gamma) \in C\}$ 
19:       $Strategies[v, w] := \gamma_{min}$ 
20:      if  $v$  nije koren 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  $w$  nije koren 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:    $Table[parent(v), w] \stackrel{\pm}{=} \begin{cases} Table[v, w] & \text{ak } v \in \gamma \\ c_{min} & \text{v opacnom pripade} \end{cases}$ 

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

```

Algorithm 4 Počítanie mapovania

```
1: procedure MAPPING( $F, G, TreeDistance$ )
2:    $\sigma \leftarrow$  lubovolna LRH strategia
3:    $ForestDistance \leftarrow$  SINGLE PATH( $F, G, TreeDistance, \sigma$ )
4:   while  $F \neq \emptyset \wedge G \neq \emptyset$  do
5:      $v \leftarrow$  UPDATE( $F, \sigma$ )
6:      $w \leftarrow$  UPDATE( $G, \sigma$ )
7:     if  $ForestDistance[F, G] = ForestDistance[F - v, G] + C_{del}$  then
8:        $Mapping \leftarrow Mapping \cup (v \rightarrow 0)$ 
9:        $F \leftarrow F - v$ 
10:    else if  $ForestDistance[F, G] = ForestDistance[F, G - w] + C_{ins}$  then
11:       $Mapping \leftarrow Mapping \cup (0 \rightarrow w)$ 
12:       $G \leftarrow G - w$ 
13:    else
14:      if  $F$  a  $G$  su stromy then
15:         $Mapping \leftarrow Mapping \cup (v \rightarrow w)$ 
16:         $F \leftarrow F - v$ 
17:         $G \leftarrow G - w$ 
18:      else
19:         $Mapping \leftarrow Mapping \cup$ 
20:          MAPPING( $F - F_v, G - G_w, TreeDistance$ )
21:         $F \leftarrow F - F_v$ 
22:         $G \leftarrow G - G_w$ 

23: procedure UPDATE( $Forest, \sigma$ )
24:    $\gamma \leftarrow$  cesta v lese  $Forest$  podľa strategie  $\sigma$ 
25:   return vrchol  $r_L(Forest)$  alebo  $r_R(Forest)$  alebo  $\emptyset$  z  $Forest$ 
26:   rovnako ako v definícii 7
```

3. Kreslenie molekuly

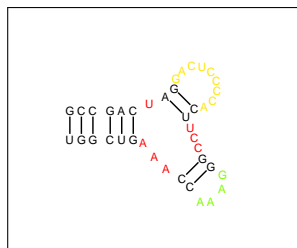
Po tom čo získame a aplikujeme mapovanie medzi šablonovou a cieľovou molekulou RNA, získame cieľovú molekulu s čiastočnou vizualizáciou, ktorej zvyšok treba dopočítať.

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 1.2. 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: 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 bazovými párami a vyrovňovanie stémov.

3.2.1 Normalizácia vzdialeností v bazových pároch a vyrovňovanie stémov

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

Algoritmus normalizácie vzdialeností medzi vrcholmi bazových 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.

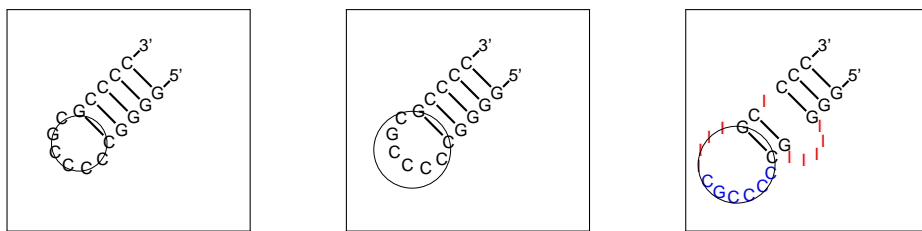
Algorithm 5 Rozloženie báz na kružnicu

```
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 pozícií 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 zmazali, 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 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 rozťahnúť, alebo naopak priblížiť k sebe.

Ak sa jedná o pridanie/odoberanie celej vetvy stromu, modifikácií sa nevyhneme. V tom prípade potrebujeme roz distribuovať 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 ??.

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

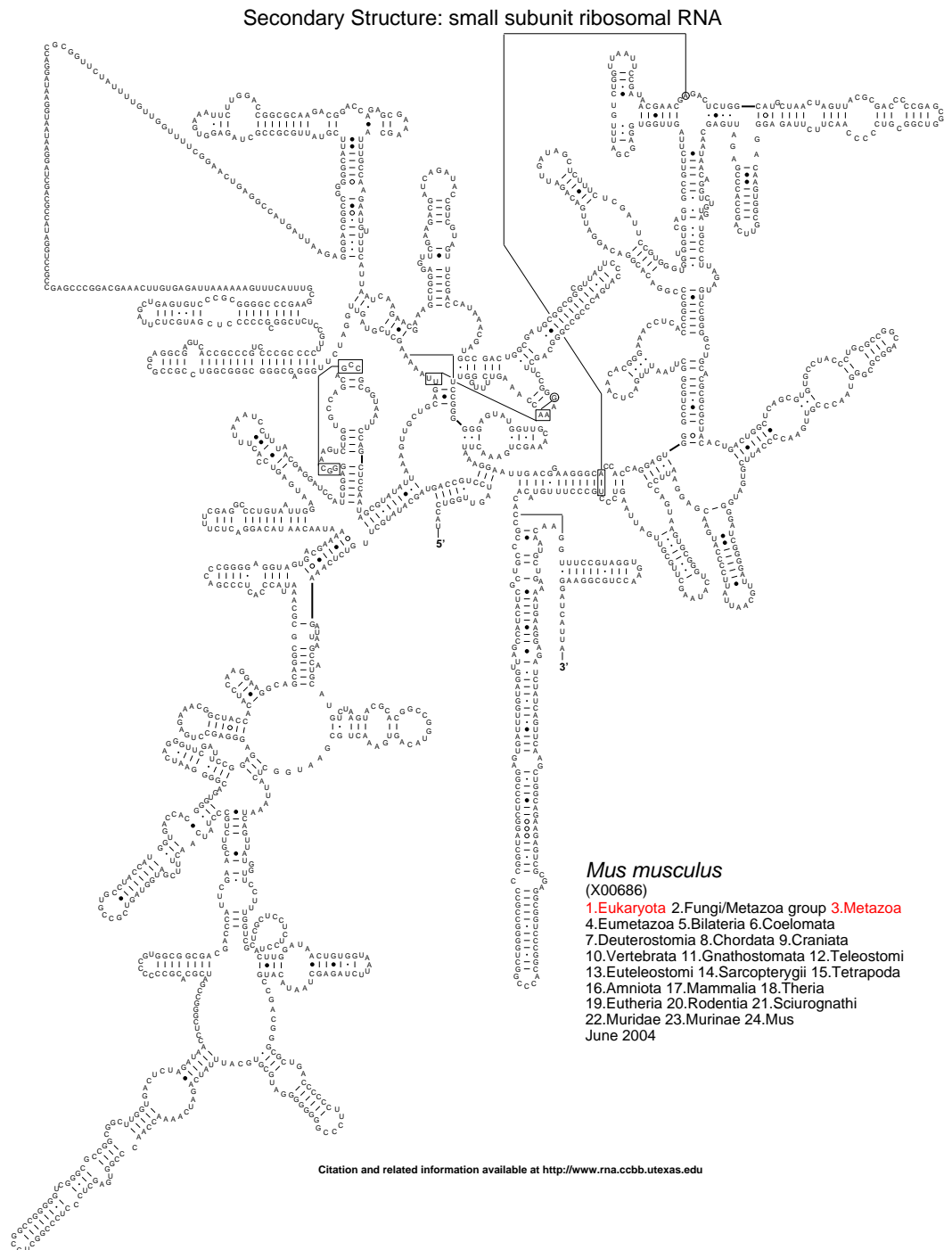
Pri tvorbe aplikácie sme už mysleli na budúcnosť a načítavanie súboru robíme v jednom ľahko rozširiteľnom module. Ten sa na základe typu v parametre *--template - tree* rozhoduje aký extractor použiť. Predvolený a jediný implementovaný je PostScript extractor fungujúci nad súbormi z CRW databázy.

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.

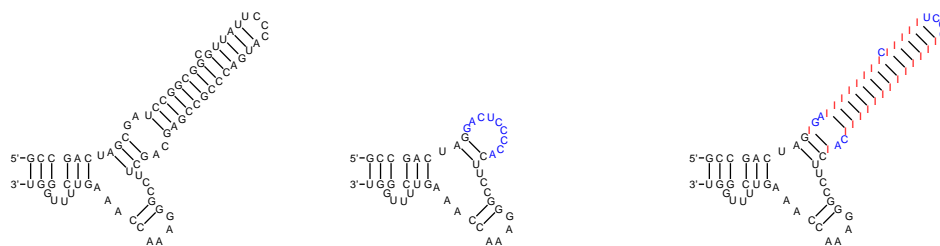
Už v predchádzajúcich kapitolách sme sa stretli s niekoľkými príkladmi, napríklad ako si program poradil s vkladaím do hairpinu - obrázok 3.2.

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

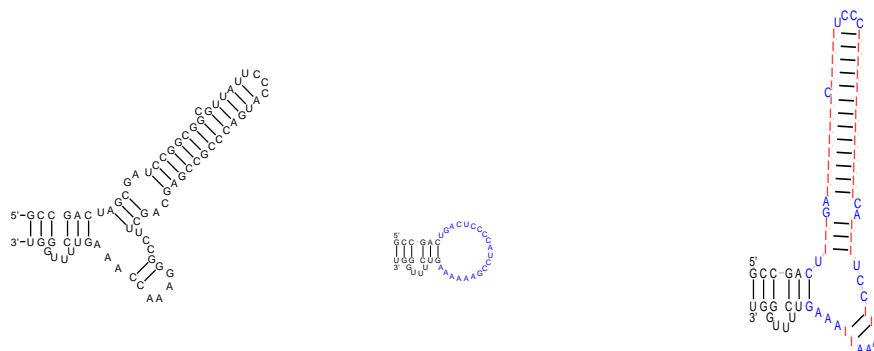
Obrázok má za cieľ ukázať, že vieme znovu nakresliť pôvodnú štruktúru iba s malými zmenami v pozícií nukleotidov (výsledne loopy sú trochu plytšie ako pôvodne).

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 ukážeme neskôr problémy nastávajú.



Obr. 5.1: Inverzne operácie: rekonštrukcia stemu

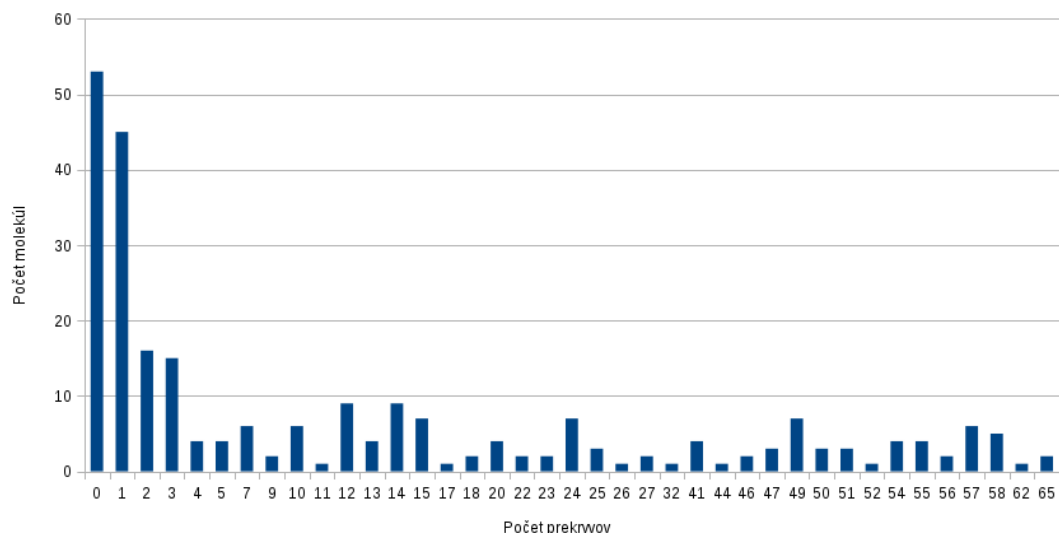


Obr. 5.2: Inverzné operácie: rekonštrukcia multibranch loop

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*.

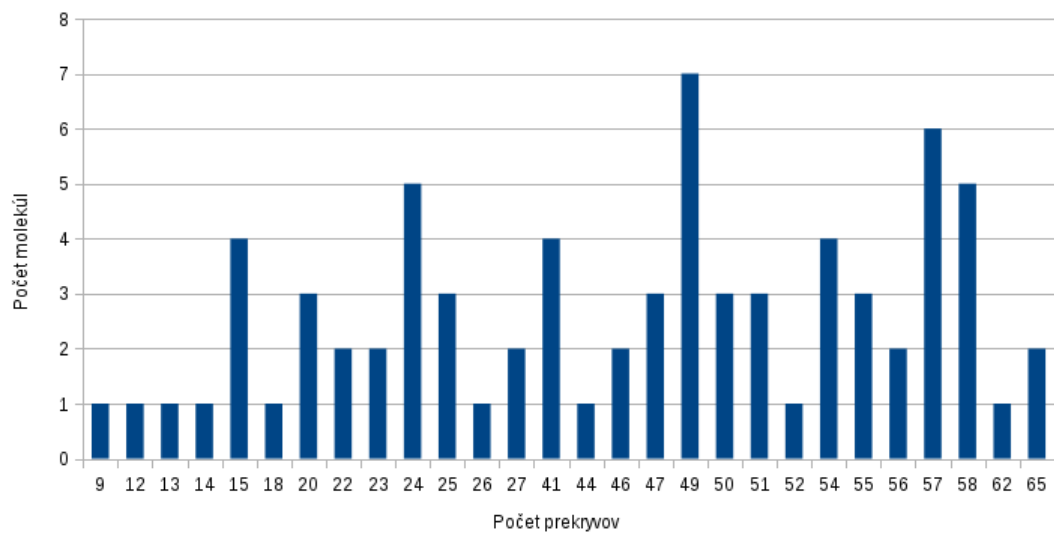
Zaujímavosťou je, že ako najvzdialenejšiu molekulu (v poradí pätnástu) si všetci vybrali molekulu od jedného konkrétneho zástupcu *echinococcus_granulosus* a vzdialenosť je 805,63 s odchylkou 12,92.

5.1 Celkové výsledky

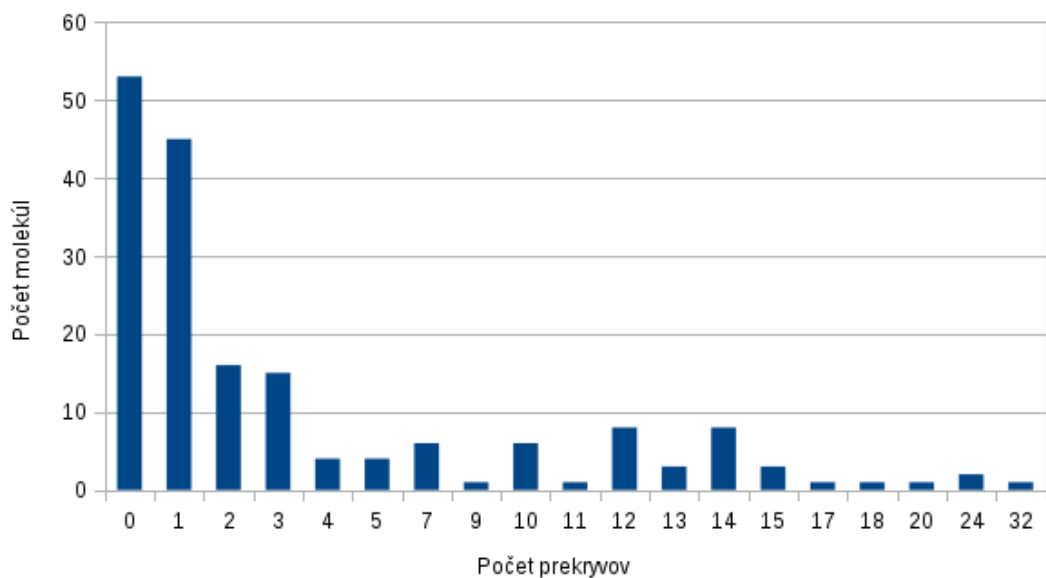
V tejto kapitole uvedieme vygenerované obrázky niektorých molekúl a na nich ukážeme časté problémy, ktoré pri vizualizácii nastávali.

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

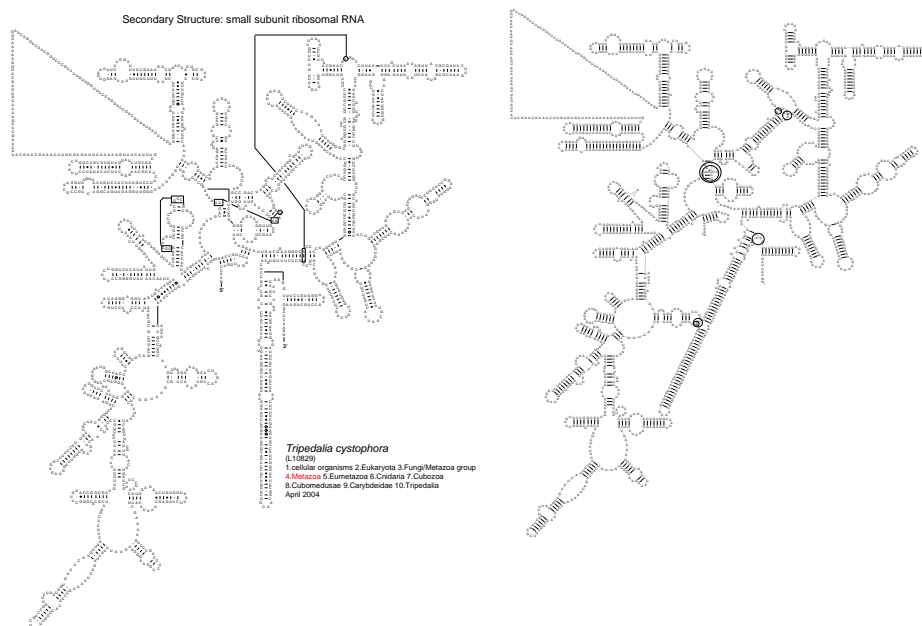
Tabuľka 5.1: Počty prekryvov v závislosti od tree-edit-distance vzdialenosti



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.

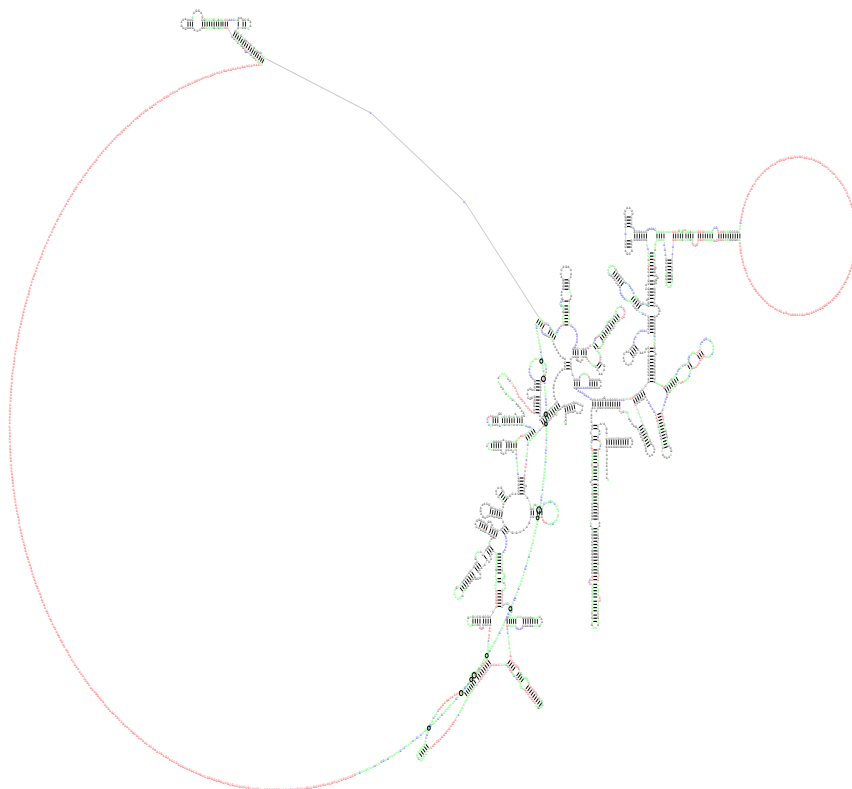
Keďže nás program všetky vzdialenosti normalizuje a následne ukladá bázy stemu na jednu priamku, vznikajú obrázky podobné 5.6.

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

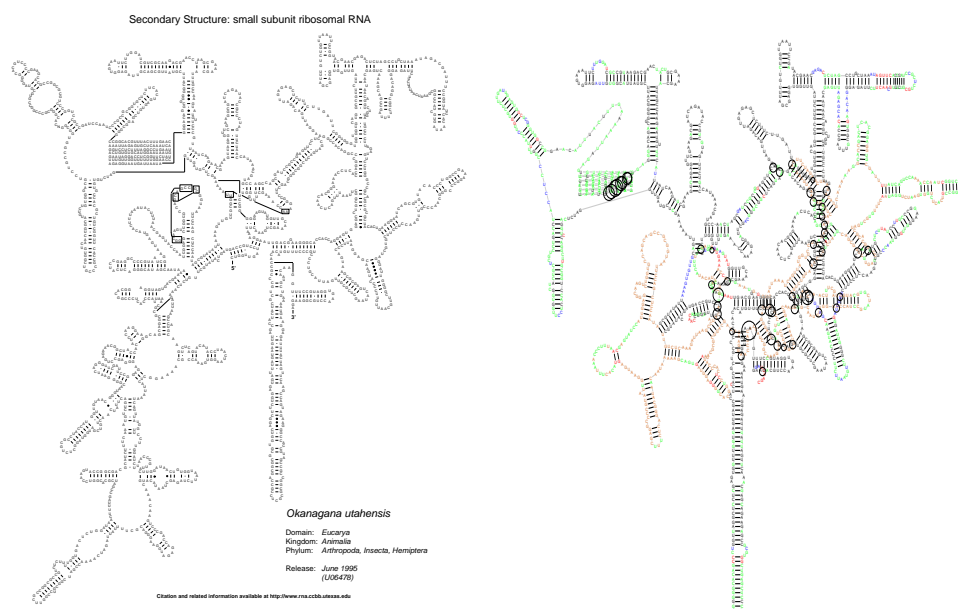
5.1.3 Otáčanie vetvy kvôli prekreslovaniu multibranch loopy

Ako už aj graf na obrázku ?? ukázal, prekreslovanie 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>.
- 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

1.1	Circular Feynman - kruhová reprezentácia sekundárnej štruktúry .	5
1.2	Strukturalne motívy v RNA	6
1.3	Varianty reprezentácie vrcholov	7
2.1	Ukázky TED operácii	8
2.2	Rekurzívny vzorec pre výpočet tree-edit-distance	9
2.3	Celková dekompozícia pomocou LRH stratégie	13
3.1	Rozlíšenie stémov a loopov v molekule: čierne sú stemy, farebne odlisene sú bázy patriace do jednej loopy	17
3.2	Príklad zvacsovania kružnice a insertu do hairpinu	19
4.1	Príklad fasta súboru	23
4.2	Príklad podporovaného formátu post script súboru	23
4.3	Príklad vstupného obrázka	25
4.4	Format výstupného PostScript súboru	26
4.5	Formát výstupného SVG súboru	26
5.1	Inverzne operácie: rekonštrukcia stému	27
5.2	Inverzné operácie: rekonštrukcia multibranch loop	27
5.3	Počet prekryvov v testovaných molekulách	28
5.4	Počet prekryvov: molekuly, ktoré potrebovali prekreslit multib- ranch loop	29
5.5	Počet prekryvov: molekuly bez prekresľovania multibranch loop .	29
5.6	Chyba pri otočení vetvy	30
5.7	Chyba pri rozkladaní báz na kružnicu	31
5.8	Chyba pri otáčaní kvôli prekresľovaniu multibranch loopy	32

Zoznam tabuliek

5.1	Počty prekryvov v závislosti od tree-edit-distance vzdialenosti . . .	28
-----	---	----

Seznam použitých zkratek

Přílohy