Univerzita Komenského, Bratislava
Fakulta Matematiky, Fyziky a Informatiky

Zarovnávanie sekvencií s použitím metód klasifikácie

Diplomová práca

2014 **Bc. Michal Hozza**

Univerzita Komenského, Bratislava
Fakulta Matematiky, Fyziky a Informatiky

Zarovnávanie sekvencií s použitím metód klasifikácie

Diplomová práca

Študijný program:	Informatika
Študijný odbor:	2508 Informatika
Školiace pracovisko:	Katedra Informatiky
Školiteľ:	Mgr. Tomáš Vinař, PhD.
Konzultant:	Mgr. Michal Nánási

Bratislava, 2014 **Bc. Michal Hozza**

Poďakovanie...

Bc. Michal Hozza

Abstrakt

Zarovnávanie dvoch DNA sekvencií je jedným zo základných bioinformatických problémov. Obvykle takéto zarovnanie hľadáme pomocou jednoduchých párových skrytých Markovovských modelov (pHMM). V tejto práci sa zaoberáme možnosťami použitia prídavnej informácie o funkcii vstupných sekvencií na zlepšenie kvality takýchto zarovnaní. Informácie sme zakomponovali pomocou klasifikátorov, ktoré rozhodujú či dané pozície majú byť zarovnané k sebe alebo nie. Ako klasifikátor sme použili Random forest.

Ukázalo sa, že klasifikátor sa dokáže naučiť, ktoré okná majú byť zarovnané k sebe a ktoré nie.

Vyvinuli sme 2 modely pre zarovnanie sekvencií s anotáciami za pomoci klasifikátora, ktoré sú založené na párových skrytých Markovovských modeloch.

Model s klasifikátorom ako emisiou, kde sme nahradili emisné tabuľky stavov výstupom z klasifikátora.

Model s klasifikátorovou páskou, kde navyše modelujeme aj výstup z klasifikátora.

Kľúčové slová: zarovnávanie sekvencií, strojové učenie, Random forest, anotácie

Abstract

English abstract

Key words: ...

Obsah

Úvod **1**

1 Zarovnávanie sekvencií **3**

1.1 Podobnosť sekvencií, sekvenčná homológia a zarovnanie 3

1.2 Párové zarovnávanie 3

1.3 Typy zarovnaní 5

1.4 Skórovacie systémy 5

1.4.1 Skórovacie matice 6

1.5 Algoritmy na hľadanie zarovnaní 6

1.5.1 Algoritmus pre globálne zarovnanie: Needelman-Wunch 7

1.5.2 Algoritmus pre lokálne zarovnanie: Smith-Waterman 9

1.5.3 Afínne skórovanie medzier 9

1.6 Zarovnávanie pomocou skrytých Markovovských modelov 11

1.6.1 Skryté markovovské modely (HMM) 11

1.6.2 Zarovnávanie pomocou párových skrytých markovovských modelov (pHMM) 12

1.6.3 Viterbiho algoritmus na pHMM 12

1.6.4 Nastavenie parametrov pHMM 14

1.7 Štatistická významnosť zarovnania 14

2 Súvisiaca práca **17**

3 Klasifikácia na základe lokálnej informácie **19**

3.1 Náhodné lesy 19

3.1.1 Klasifikačné stromy 20

3.1.2 Náhodný les 22

3.2 Použitie náhodných lesov na klasifikáciu zarovnaní 23
3.2.1 Výber atribútov 23
3.2.2 Trénovanie a testovanie klasifikátorov 25
3.3 Výsledky experimentov 27
3.3.1 Dôležitosť atribútov 27
3.3.2 Úspešnosť klasifikátora 30
3.4 Zhrnutie 31

4 Modely **32**
4.1 Model s klasifikátorom ako emisiou (Model A) 32
4.2 Model s klasifikátorovou páskou (Model B) 33
4.2.1 Diskrétna verzia 33
4.2.2 Spojitá verzia 34
4.3 Trénovanie modelov 35
4.3.1 Trénovanie modelu A 35
4.3.2 Trénovanie modelu B 35
4.3.3 Popis trénovacej a testovacej množíny 36
4.4 Výsledky experimentov 36
4.4.1 Porovnanie modelov 36
4.4.2 Rôzne veľkosti okna 38
4.4.3 Vplyv dodatočnej informácie na zarovnanie 39
4.4.4 Použitie jedného klasifikátora pre Match aj Inzert stav 39
4.4.5 Porovnanie modelov s existujúcimi zarovnávačmi 40
4.5 Zhrnutie 40

5 Implementácia **41**
5.1 Použité knižnice 41
5.1.1 Realigner 41
5.1.2 Pythonové knižnice 41
5.2 Triedy na zarovnávanie s klasifikátorom 41
5.2.1 Predspracovanie dát 41
5.2.2 Zovšeobecnenie klasifikátora 41
5.2.3 HMM stavy s klasifikátorom 41

5.3 Pomocné programy 42
5.3.1 Simulátor 42
5.3.2 Trénovanie modelov 43
5.3.3 Testovanie klasifikátora 44
5.4 Použitie 44

Záver **45**

Literatúra **46**

Zoznam obrázkov

1.1 Lokálne zarovnanie 4
1.2 Skórovacia matica 6
1.3 Tabuľka dyn. programovania pre globálne zarovnanie 7
1.4 Tabuľka dyn. programovania pre lokálne zarovnanie 9
1.5 Situácie pri afínnoom skórovaní 10
1.6 Stavový diagram pre zarovnanie sekvencií 11
1.7 Párový HMM pre zarovnávanie sekvencií 13
1.8 P-hodnota lokálneho zarovnania 15

3.1 Klasifikačný strom 21
3.2 Okno klasifikátora 24
3.3 Dôležitosť atribútov pre typ dát D 29
3.4 Distribúcia výstupu z klasifikátora pri type D 30
3.5 Horšie distribúcie výstupov z klasifikátora 31

4.1 Model s klasifikátorom ako emisiou 32
4.2 Model s klasifikátorovou páskou 33
4.3 Diskretizácia výstupu klasifikátora 34

5.1 Markovova reťaz použitá na generovanie informácie o génoch 42

Zoznam tabuliek

3.1 Porovnanie vlastností klasifikátorov 28

4.1 Porovnanie úspešností pri rôznom spracovaní pásky 35
4.2 Pravdepodobnosť mutácie v našich datasetoch 37
4.3 Porovnanie emisných pravdepodobností 37
4.4 Porovnanie úspešností modelov 38
4.5 Porovnanie úspešností pri rôznej veľkosti okna 38
4.6 Vplyv dodatočnej informácie na zarovnanie 39
4.7 Porovnanie použitia jedného alebo dvoch typov klasifikátorov 39
4.8 Porovnanie s existujúcimi zarovnávačmi 40

Úvod

Najnovšie technológie sekvenovania DNA produkujú stále väčšie množstvo sekvencií rôznych organizmov. Spolu s tým stúpa aj potreba rozumieť týmto dátam. Dôležitým krokom k ich porozumeniu je *zarovnávanie sekvencií*. Zarovnávanie dvoch DNA sekvencií je teda jedným zo základných bioinformatických problémov. Správne zarovnanie identifikuje časti sekvencie, ktoré vznikli z toho istého predka (zarovnané bázy), ako aj inzercie a delécie v priebehu evolúcie (medzery v zarovnaní). Je nápomocné pri zisťovaní ich štruktúry a následne funkcie jednotlivých častí.

Existujú rôzne algoritmy na zarovnávanie sekvencií. Väčšina z nich je založená na pravdepodobnostnom modeli, pričom sa snažia nájsť zarovnanie s čo najväčšou pravdepodobnosťou. Algoritmy sú zvyčajne založené na dynamickom programovaní a pracujú v kvadratickom čase v závislosti od dĺžok sekvencií. Niekedy sa na urýchlenie použijú rôzne heuristické algoritmy, ktoré nie vždy nájdu najpravdepodobnejšie zarovnanie, ale pracujú oveľa rýchlejšie.

My sme sa v práci zaoberali algoritmom, ktorý hľadá zarovnanie pomocou jednoduchých párových skrytých Markovovských modelov (pHMM) [DEKM98], kde kvalita výsledného zarovnania je ovplyvnená len pravdepodobnostným modelom.

Základný model berie do úvahy len jednotlivé *bázy* a pravdepodobnosti *substitúcie (mutácie)*, *inzercie* a *delécie*. Náš model navyše uvažuje aj prídavné informácie (takzvané anotácie) získané z externých programov (napr. anotácie o génoch z vyhľadávača génov).

Keďže množstvo dodatočnej informácie môže byť veľmi veľké – napríklad pre 3 binárne anotácie by sme mali $2^3 \times 2^3 = 64$ krát väčší počet parametrov – je ťažké skonštruovať vhodnú skórovaciu maticu pre zarovnávací algoritmus. Namiesto nej sme teda použili klasifikátory[1], ktoré sme trénovali na sekvenciách so známym zarovna-

[1]program, ktorý na základe vstupnej informácie a vopred natrénovaných parametrov klasifikuje dáta

ním a potom použili na zarovnanie nových sekvencií. Naše klasifikátory vracajú čísla z intervalu $\langle 0, 1 \rangle$, ktoré určujú, či dané dve bázy majú byť zarovnané spolu.

Ako klasifikátor sme použili *náhodný les* [Bre01], pretože aktuálne patrí medzi najlepšie klasifikátory.

ToDo: napisat este strucny obsah prace

do niektorej triedy z danej množiny tried

1 Zarovnávanie sekvencií

V tejto kapitole si stručne popíšeme čo je to globálne a lokálne zarovnanie a ukážeme základné algoritmy na hľadanie globálneho a lokálneho zarovnania. Tieto algoritmy budeme neskôr modifikované používať pri našom riešení.

1.1 Podobnosť sekvencií, sekvenčná homológia a zarovnanie

V prírode vznikajú evolúciou nové sekvencie modifikáciou už existujúcich. Preto môžme často spozorovať podobnosť medzi neznámou sekvenciou a sekvenciou o ktorej už niečo vieme. Ak zistíme podobnosti medzi sekvenciami, môžeme preniesť informácie o štruktúre a/alebo funkcii na novú sekvenciu.

Podobné sekvencie, ktoré sa vyvinuli mutáciami zo sekvencie v spoločnom predkovi sa nazývajú *homologické* a pod pojmom *hľadanie homológov* rozumieme hľadanie takých podobností, ktoré s veľkou pravdepodobnosťou vznikli práve spoločnou evolučnou históriou.

Počas evolúcie dvoch homologických sekvencií nastane veľa *inzercií*, *delécií* a *substitúcií*, preto predtým ako môžeme začať porovnávať sekvencie, ich musíme zarovnať tak, aby homologické časti sekvencií boli na rovnakom mieste v zarovnaní. [DEKM98, BV11]

1.2 Párové zarovnávanie

Párové zarovnávanie je základná úloha zarovnávania sekvencií, kde sa k sebe zarovnávajú dve sekvencie. V tejto práci sa budeme zaoberať len párovým zarovnávaním.

Kľúčové problémy sú:

Sekvencia 1:

gagacccgcctaggtgaatatttagcagc
gattaaataccacgtaTATAAGGTGGACC
GTTCCTCGAGAGGTTCTTCCGGCAATGAC
GGCCAGAGCAAAAGCCACGTgtaggactg
catacgcctctacgcctccactgacgcga
tgatgtggcgtggatctgtttgctcttgg
tataggtcacggagacggctggtactgat
cccttcgggagtaaaaatataatgaccat
ggcccaggcttcaggagggagttgtgcgg

Sekvencia 2:

tgtacagcactgcaacgagcatctggggg
ttggttattccgatggcgctggacagcta
gcggacagtagttctcaggccttagtaga
aaggtgggaaccccctATGAGGTCGACCG
TTTCAGCGTGACTATAGACGTCATTGAAG
CAATATACAGGAACACCACCTacttagga
agggagttcggtgcagtaaagcattctta
cctcagggcacggtagagaacactacaac
cagaatagcaacgtgatgcggcgactctc

Lokálne zarovnanie:

```
TATAAGGTGGACCGTT--------CCTCGAGAGGTTCTTCCGGCAATGGCCACGAGAGCAAAAGCCACGT
TATGAGGTCGACCGTTTCAGCGTGACTATAGACGTCATTGAAGCAATATACAGG------AACACCACCT
```

Obr. 1.1: Dve sekvencie a ich lokálne zarovnanie. Veľkými písmenami a červenou farbou sú vyznačené zarovnané časti. V zarovnaní sa nachádzajú zhody, nezhody a medzery v oboch sekvenciách

1. Aké typy zarovnávania by sme mali uvažovať
2. Skórovací systém, ktorý použijeme na ohodnotenie zarovnania a trénovanie
3. Algoritmus, ktorý použijeme na hľadanie optimálneho alebo dobrého zarovnania podľa skórovacieho systému
4. Štatistická významnosť zarovnania.

[DEKM98]

1.3 Typy zarovnaní

Základné typy zarovnaní sú *Globálne zarovnanie* a *Lokálne zarovnanie*.

Definícia 1.3.1 (Globálne zarovnanie)**.** Vstupom sú dve sekvencie $X = x_1x_2 \dots x_n$ a $Y = y_1y_2 \dots y_m$ Výstupom je zarovnanie celých sekvencií X a Y.

Definícia 1.3.2 (Lokálne zarovnanie)**.** Vstupom sú dve sekvencie $X = x_1x_2 \dots x_n$ a $Y = y_1y_2 \dots y_m$ Výstupom je zarovnanie nejakých podreťazcov $x_i \dots x_j$ a $y_k \dots y_l$ sekvencií.

V praxi sa väčšinou snažíme nájsť zarovnania s najvyšším skóre. [BV11]

1.4 Skórovacie systémy

Takmer všetky metódy zarovnania hľadajú zarovnanie dvoch reťazcov na základe nejakej *skórovacej schémy*[1]. Skórovacie schémy môžu byť veľmi jednoduché, napr. $+1$ za *zhodu* a -1 za *nezhodu*. Hoci ak chceme mať schému, kde biologicky najkorektnejšie zarovnanie má najvyššie skóre, musíme vziať do úvahy, že biologické sekvencie majú evolučnú históriu, 3D štruktúru a mnohé ďalšie vlastnosti obmedzujúce ich evolučné procesy. Preto skórovací systém vyžaduje starostlivé premyslenie a môže byť veľmi zložitý. [DEKM98]

[1] metóda, ktorou priradíme zarovnaniu skóre - zvyčajne čím väčšie skóre, tým realistickejšie zarovnanie by to malo byť

1.4.1 Skórovacie matice

Skoro vždy však chceme rôzne zhody a nezhody skórovať rôzne - nie len všetky zhody $+1$ a nezhody -1. Skóre môže záviseť od toho aké bázy sú v danom stĺpci zarovnania. Na to sa používa *skórovacia matica* (obr. 1.2), kde máme definované skóre pre každú dvojicu. Skórovacie matice sa využívajú najmä pri zarovnávaní proteínov, kde niektoré dvojice majú podobné chemické vlastnosti. [DEKM98, BV11]

Iným typom skórovacej schémy je napríklad *párový skrytý markvovský model* (viac v 1.6)

	A	C	G	T	-
A	6	-1	-2	-1	-3
C	-1	5	-3	-2	-4
G	-2	-3	5	-2	-2
T	-1	-2	-2	6	-1
-	-3	-4	-2	-1	∞

Obr. 1.2: Ukážka skórovacej matice. Všimnime si, že môžu byť rôzne skóre aj za jednotlivé zhody, nezhody alebo medzery.

1.5 Algoritmy na hľadanie zarovnaní

Pre danú skórovaciu schému potrebujeme algoritmus, ktorý nájde optimálne zarovnanie dvoch sekvencií. Budeme uvažovať zarovnávanie s medzerami. Medzery používame na znázornenie delécie v danej sekvencii, alebo inzercie v druhej sekvencii. Na označenie medzier používame pomlčku `'-'` a medzeru do sekvencie medzi 2 bázy pridáme tak, že medzi 2 bázy napíšeme jednu alebo viac pomlčiek - počet pomlčiek zodpovedá počtu medzier a teda aj počtu báz, ktoré má na danom mieste jedna sekvencia navyše oproti druhej. Do sekvencie môžeme pridať ľubovoľne veľa medzier, aby sme dosiahli lepšie skóre. Pre 2 sekvencie dĺžky n existuje

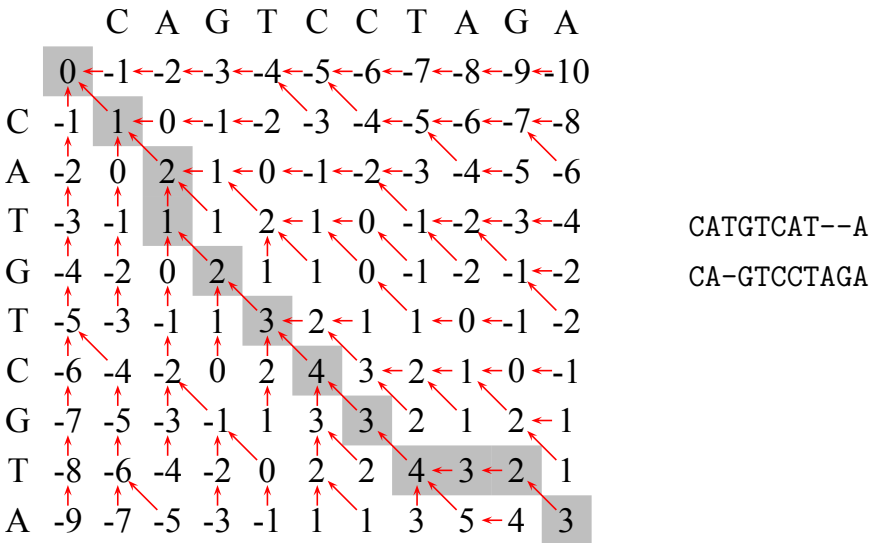
$$\binom{2n}{n} = \frac{(2n)!}{(n!)^2} \simeq \frac{2^{2n}}{\sqrt{\pi n}}$$

možných globálnych zarovnaní. [DEKM98] Čiže nie je možné v rozumnom čase nimi prejsť.

Algoritmy na hľadanie zarovnaní využívajú *dynamické programovanie*. Často sa používajú aj rôzne heuristiky na urýchlenie výpočtu. My sa budeme zaoberať len algoritmami využívajúcimi dynamické programovanie. Pre rôzne typy zarovnaní a skórovacie schémy máme rôzne algoritmy zarovnávania. [DEKM98, BV11]

1.5.1 Algoritmus pre globálne zarovnanie: Needelman-Wunch

Máme dané 2 sekvencie $X = x_1x_2 \dots x_n$ a $Y = y_1y_2 \dots y_m$, budeme zarovnávať všetky znaky sekvencie X a všetky znaky sekvencie Y. Definujeme si jednoduchú skórovaciu tabuľku kde $s(x, y)$ bude udávať skóre pre danú dvojicu báz (napr. $+1$ za zhodu, -1 za nezhodu) a za nezhodu budeme dávať penaltu $-d$.



Obr. 1.3: Tabuľka dynamického programovania pre globálne zarovnanie (vľavo) a výsledné zarovnanie (vpravo). Skóre je +1 za zhodu a -1 za nezhodu alebo medzeru.

Algoritmus postupne vypĺňa 2-rozmernú maticu A. Riadky zodpovedajú bázam sekvencie X a stĺpce bázam Y. Na políčku $A[i, j]$ bude skóre najlepšieho zarovnania prvých i báz sekvencie X a prvých j báz Y.

Keď zarovnávame sekvenciu s prázdnou sekvenciou, tak skóre bude $-n$, kde n je dĺžka

sekvencie. Bude tam n pomlčiek, každá nám dá skóre -1. Takto vyplníme riadky a stĺpce $A[i,0]$ a $A[0,j]$.

Ak chceme vyplniť políčko $A[i,j]$, musíme si uvedomiť ako môže vyzerať posledný stĺpec zarovnania $x_1x_2\ldots x_i$ a $y_1y_2\ldots y_j$. Máme iba 3 možnosti ako môže vyzerať posledný stĺpec najlepšieho zarovnania. Buď obsahuje x_i alebo y_j alebo oboje. V prípade, že posledný stĺpec obsahuje oboje, cena tohto stĺpca je $s(x_i,y_j)$. Ak by sme posledný stĺpec zmazali, dostali by sme zarovnanie $x_1x_2\ldots x_{i-1}$ a $y_1y_2\ldots y_{j-1}$, pričom musí ísť o najlepšie zarovnanie. To už máme vypočítané v políčku $A[i-1,j-1]$, čiže výsledné skóre bude $A[i-1,j-1]+s(x_i,y_j)$.

V prípade, že posledný stĺpec obsahuje len x_i zarovnané s pomlčkou, skóre stĺpca bude -1 a po zmazaní dostávame zarovnanie $x_1x_2\ldots x_{i-1}$ a $y_1y_2\ldots y_j$, výsledné skóre bude teda $A[i-1,j]-1$. V prípade, že posledný stĺpec obsahuje len y_i, tak skóre vypočítame analogicky.

Najlepšie skóre bude maximálne skóre pre všetky 3 prípady. Dostávame teda nasledujúci vzťah pre výpočet $A[i,j]$:

$$A[i,j]=\max\begin{cases}A[i-1,j-1]+s(x_i,y_j)\\A[i-1,j]-d\\A[i,j-1]-d\end{cases}$$

Maticu vieme vypĺňať po riadkoch, pričom každé políčok vieme vypočítať z troch políčok, ktoré už sú vypočítané. Políčko $A[0,0]=0$ a krajné políčka potom vieme vypočítať ako $A[i,0]=A[i-1,j]-d$ a $A[0,j]=A[i,j-1]-d$

Ak nás zaujíma aj zarovnanie – nie len jeho skóre – vieme si pre každé políčko zapamätať ktorá z troch možností dosiahla maximálnu hodnotu (červené šípky na Obr. 1.3). Na základe tejto informácie potom vieme zrekonštruovať zarovnanie tak, že postupne z posledného políčka ($A[n,m]$) budeme prechádzať na políčko, z ktorého sme vypočítali aktuálnu hodnotu.

Časová zložitosť je $O(nm)$, pretože vypĺňame nm políčok, každé v konštantnom čase. Zjavne aj pamäťová zložitosť je $O(nm)$.

Pamäťová zložitosť sa dá zredukovať na $O(n+m)$ za cenu zhruba dvojnásobného času výpočtu [Hir75].

1.5.2 Algoritmus pre lokálne zarovnanie: Smith-Waterman

		C	A	G	T	C	C	T	A	G	A
	0	0	0	0	0	0	0	0	0	0	0
A	0	0	1	0	0	0	0	0	1	0	1
T	0	0	0	0	1	0	0	1	0	0	0
G	0	0	0	1	0	0	0	0	0	1	0
T	0	0	0	0	2	1	0	1	0	0	0
C	0	1	0	0	1	3	2	1	0	0	0
T	0	0	0	0	1	2	2	3	2	1	0
A	0	0	1	0	0	1	1	2	4	3	2
T	0	0	0	0	1	0	0	2	3	3	2

```
GT-CTA
GTCCTA
```

Obr. 1.4: Tabuľka dynamického programovania pre lokálne zarovnanie (vľavo) a výsledné zarovnanie (vpravo). Skóre je +1 za zhodu a -1 za nezhodu alebo medzeru.

Algoritmus pre lokálne zarovnania sa líši len v niekoľkých malých detailoch. Opäť vypĺňame maticu A, s tým, že v $A[i, j]$ bude najvyššie skóre lokálneho zarovnania medzi sekvenciami $x_1x_2 \ldots x_i$ a $y_1y_2 \ldots y_j$, ktoré buď obsahuje bázy x_i aj y_j, alebo je prázdne. Teda na ľubovoľnom mieste uvažujeme aj prázdne zarovnanie so skóre 0 (v matici nebudú záporné čísla). Vzťah pre výpočet $A[i, j]$ vyzerá takto:

$$A[i,j] = \max \begin{cases} 0 \\ A[i-1, j-1] + s(x_i, y_j) \\ A[i-1, j] - d \\ A[i, j-1] - d \end{cases}$$

V tomto prípade sú všetky krajné políčka nulové.

Časová aj pamäťová zložitosť sú, rovnako ako pri globálnom zarovnaní $O(nm)$.

1.5.3 Afínne skórovanie medzier

V jednoduchom skórovaní sme dávali za pomlčku vždy rovnaké skóre (-1). Pri evolúcii sa však môže stať, že sa naraz zmaže niekoľko susedných báz. Pri *afínnom skórovaní*

medzier teda zavedieme dva typy skóre. Skóre za *začatie medzery* a skóre za *rozšírenie medzery.*

Algoritmus globálneho zarovnania vieme upraviť nasledovne: Namiesto matice A teraz budeme mať 3 matice M, I_x, I_y zodpovedajúce trom situáciám (Obr. 1.5).

ACTx$_i$	ACTTAx$_i$	ACTx$_i$--
AGTy$_j$	AGTy$_j$--	AGTATy$_j$
(a) Mutácia(M)	(b) Inzercia v X (I_x)	(c) Inzercia v Y (I_y)

Obr. 1.5: Tri situácie pri afínnoom skórovaní medzier

Nech $M[i,j]$ je najlepšie skóre prvých i báz zo sekvencie X a prvých j báz zo sekvencie Y, pričom x_i je zarovnané k y_j, $I_x[i,j]$ je najlepšie skóre ak x_i je zarovnané k medzere a $I_y[i,j]$ je najlepšie skóre ak y_j je zarovnané k medzere.

Označme si d penaltu za začatie medzery a e penaltu za rozšírenie medzery. Vzťahy pre výpočet políčok sú nasledovné:

$$M[i,j] = \max \begin{cases} M[i-1,j-1] + s(x_i, y_j) \\ I_x[i-1,j-1] + s(x_i, y_j) \\ I_y[i-1,j-1] + s(x_i, y_j) \end{cases}$$

$$A[i,j] = \max \begin{cases} M[i-1,j] - d \\ I_x[i-1,j] - e \end{cases}$$

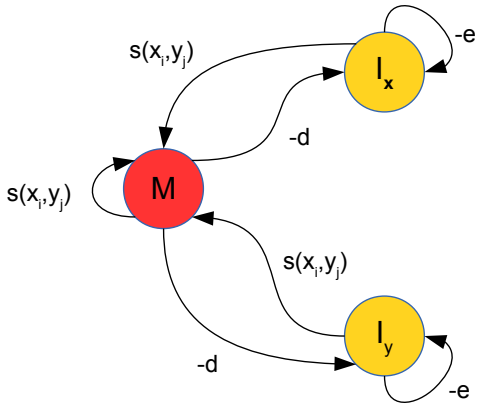
$$A[i,j] = \max \begin{cases} M[i,j-1] - d \\ I_y[i,j-1] - e \end{cases}$$

V týchto rovniciach predpokladáme, že delécia nie je nasledovaná inzerciou. Toto platí v optimálnej sekvencii, ak $-d - s$ je menšie ako najmenšie skóre nezhody.

Tieto vzťahy vieme popísať stavovým diagramom na obrázku 1.6:

Časová zložitosť je $O(nm)$, pretože vypĺňame $3nm$ políčok, každé v konštantnom čase. Pamäťová zložitosť je opäť $O(nm)$.

[DEKM98]



Obr. 1.6: Stavový diagram pre zarovnanie sekvencií - obsahuje *Match* (M), *InsertX* (I_x) a *InsertY* (I_y) stav a prechody medzi nimi spolu s ich cenou. Napríklad prechod z M do I_x znamená vloženie medzery do Y-ovej sekvencie a to penalizujeme $-d$

1.6 Zarovnávanie pomocou skrytých Markovovských modelov

1.6.1 Skryté markovovské modely (HMM)

Skrytý markovovský model (Hidden Markov Model, HMM) je pravdepodobnostný model, ktorý generuje náhodnú sekvenciu spolu s jej anotáciou (stavmi). HMM si môžme predstaviť ako konečný automat. Skladá sa z niekoľkých stavov, prechodov medzi nimi a emisií. Na rozdiel od bežných konečných automatov, HMM emitujú symboly v stave, nie počas prechodu. HMM sa skladá z 3 distribúcií

- distribúcia začiatočných stavov (HMM začne v stave i)
- distribúcia prechodov (HMM prejde zo stavu i do stavu j)
- distribúcia emisií (HMM v stave i vygeneruje symbol x)

Generovanie sekvencie teda vyzerá nasledovne: Na začiatku je HMM v niektorom stave (každý stav i má nejakú pravdepodobnosť π_i, že bude začiatočný). Potom v každom kroku HMM emituje symbol x s pravdepodobnosťou $e_{i,x}$ a prejde do stavu j s pravdepodobnosťou $a_{i,j}$. Po n krokoch takto vygenerujeme sekvenciu dĺžky n, pričom každý symbol je oanotovaný stavom, ktorý ho vygeneroval.

V takomto modeli vieme počítať pravdepodobnosť, že model vygeneruje sekvenciu x dĺžky n s anotáciou s ako súčin pravdepodobností prechodov a emisií. Výpočet vyzerá nasledovne:

$$P[X = x|S = s] = \pi_{s_1} e_{s_1,x1} a_{s_1,s_2} e_{s_2,x_2} a_{s_2,s_3} e_{s_3,x_3} \dots a_{s_{n-1},s_n} e_{s_n,x_n}$$

. [BV11, DEKM98]

1.6.2 Zarovnávanie pomocou párových skrytých markovovských modelov (pHMM)

Párové skryté Markovovské modely (pHMM) sa od tých obyčajných líšia v tom, že namiesto jednej sekvencie generujú dvojicu sekvencií. Na pHMM sa dajú použiť podobné algoritmy ako na HMM, ale treba ich upraviť tak, aby pracovali v dvoch rozmeroch.

V časti 1.5.3 sme si ukázali jednoduchý algoritmus na globálne zarovnávanie s afínnym skórovaním medzier. K tomuto algoritmu sme si uviedli aj jednoduchý stavový automat (Obr. 1.6). Tento automat vieme previesť na pHMM.

Na to aby sme automat previedli na pHMM, musíme urobiť niekoľko zmien – musíme nastaviť emisné a prechodové pravdepodobnosti, tak aby sčítavali do jedna. Pre jednoduchosť pridáme aj prechody medzi stavmi X a Y. Ak ich pravdepodobnosti nastavíme na nula, máme model ekvivalentný predchádzajúcemu. [DEKM98]

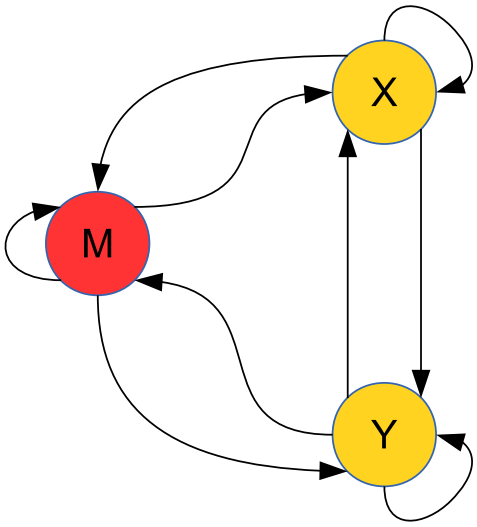
1.6.3 Viterbiho algoritmus na pHMM

Hľadáme najpravdepodobnejšiu postupnosť stavov A ak máme dané sekvencie X a Y, teda $\arg\max_A \Pr(A, X, Y)$. Úlohu budeme riešiť dynamickým programovaním.

Podproblém $V[i, j, u]$ je pravdepodobnosť najpravdepodobnejšej postupnosti stavov končiacej v $X[i]$ a $Y[j]$ v stave u.

Inicializácia:

$$V[0, 0, M] = 1 \wedge V[i, 0, *] = 0, V[0, j, *] = 0 \quad \forall i, j \tag{1.1}$$



Obr. 1.7: Párový HMM pre zarovnávanie sekvencií

Rekurentné vzťahy:

$$V[i,j,M] = e_{M,(x_i,y_j)} \max \begin{cases} a_{M,M} V[i-1,j-1,M] \\ a_{I_x,M} V[i-1,j-1,I_x] \\ a_{I_y,M} V[i-1,j-1,I_y] \end{cases} \tag{1.2}$$

$$V[i,j,I_x] = e_{I_x,x_i} \max \begin{cases} a_{M,I_x} V[i-1,j,M] \\ a_{I_x,I_x} V[i-1,j,I_x] \\ a_{I_y,I_x} V[i-1,j,I_y] \end{cases} \tag{1.3}$$

$$V[i,j,I_y] = e_{I_y,y_j} \max \begin{cases} a_{M,I_y} V[i,j-1,M] \\ a_{I_y,I_y} V[i,j-1,I_y] \\ a_{I_x,I_y} V[i,j-1,I_x] \end{cases} \tag{1.4}$$

Algoritmus funguje takto: Nech n a m sú dĺžky sekvencií X a Y. Na začiatku nainicializuje hodnoty pre $V[0,*,*]$ a $V[*,0,*]$ podľa 1.1. Potom postupne pre všetky $i = 1 \ldots n$, $j = 1 \ldots m$ a $u \in \{M, I_x, I_y\}$ dynamicky vypočíta $V[i,j,u]$ podľa 1.2, 1.3 a 1.4.

Maximálne $V[n,m,u] \forall u \in \{M, I_x, I_y\}$ je pravdepodobnosť najpravdepodobnejšej postupnosti stavov. Aby sme stavy vedeli zrekonštruovať, pamätáme si pre každé $V[i,j,u]$ stav w, ktorý viedol k maximálnej hodnote. Táto postupnosť stavov nám udáva najpravdepodobnejšie zarovnanie.

Časová zložitosť tohto algoritmu je $O(nm)$.

Poznámka: pre dlhé sekvencie budú čísla $V[i, j, u]$ veľmi malé a môže dôjsť k podtečeniu. V praxi teda používame zlogaritmované hodnoty a namiesto násobenia súčet. [DEKM98]

1.6.4 Nastavenie parametrov pHMM

Ak máme oanotované trénovacie sekvencie, môžme z nich parametre odvodiť frekvenčnou analýzou. Emisie získame tak, že vyfiltrujeme symboly s príslušným stavom a spočítame frekvencie pre každý stav zvlášť, pričom v Match stave počítame frekvencie dvojíc symbolov. Tranzície získame tak, že pre každý stav spočítame frekvencie nasledujúcich stavov. Tento postup sa volá *metóda maximálnej vierohodnosti (v angličtine maximum likelihood estimation)*. [DEKM98, Wik14a]

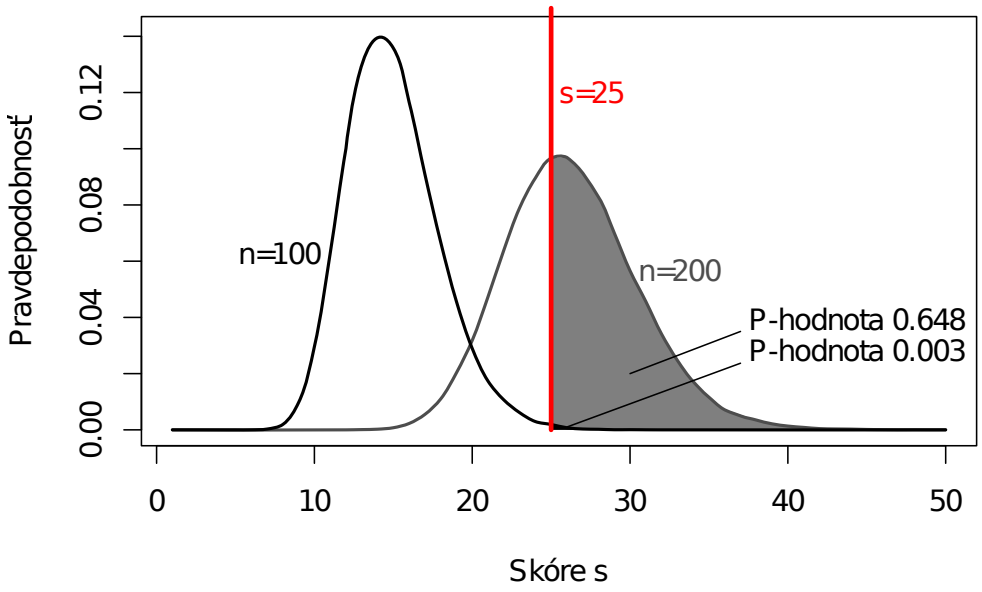
Parametre modelu môžme teda ľahko natrénovať z existujúcich párových zarovnaní.

1.7 Štatistická významnosť zarovnania

Smith-Watermanov algoritmus nájde najlepšie lokálne zarovnanie pre ľubovoľné dve sekvencie. Treba však rozhodnúť, či je zarovnanie dostatočne vierohodné nato, aby predstavovalo skutočnú podobnosť sekvencií a nie len najlepšie zarovnanie dvoch nesúvisiacich sekvencií. Ako vodítko pri rozhodnutí sa používajú identifikátory *štatistickej významnosti* zarovnania: *P-hodnota (P-value)* alebo *E-hodnota (E-value)*.

P-hodnota zarovnania je pravdepodobnosť, že medzi náhodne generovanými sekvenciami tej istej dĺžky by sme našli zarovnanie s rovnakým skóre alebo vyšším. Keďže P-hodnota závisí od dĺžok sekvencií a skóre, musíme ju počítať pri každom zarovnaní. Je však časovo náročné robiť to generovaním veľkého množstva zarovnaní, preto sa používajú matematicky odvodené vzorce na odhad tejto hodnoty. ([KA90], [MB06]).

E-hodnota vyjadruje strednú hodnotu počtu zarovnaní so skóre aspoň takým ako má naše zarovnanie medzi náhodne generovanými sekvenciami. E-hodnota teda môže byť aj väčšia ako jedna. Ak je E-hodnota väčšia ako jedna, tak čisto náhodou by sme očakávali aspoň jedno také silné zarovnanie, a teda zarovnania s takouto (a nižšou) E-hodnotou nebudeme považovať za štatisticky významné.



Obr. 1.8: P-hodnota lokálneho zarovnania so skóre $s = 25$ medzi 2 sekvenciami dĺžky $n = 100$ alebo $n = 200$ (skórovanie $+1$ zhoda, -1 nezhoda alebo medzera). Rozdelenie bolo získané zarovnávaním 100000 párov náhodných sekvencií. Pri $n = 100$ je P-hodnota približne 0.003 a zodpovedá malej čiernej ploche pod krivkou napravo od zvislej čiary pre $s = 25$. Pri dlhších sekvenciách zodpovedá P-hodnota veľkej sivej ploche napravo od zvislej čiary. Pri takto dlhých sekvenciách očakávame skóre 25 alebo väčšie vo viac ako 60% prípadov čisto náhodou. Nejde teda o štatisticky významné zarovnanie.

V štatistike sa v rôznych testoch štandardne používajú prahy na P-hodnotu 0.05 alebo 0.01. Pri zarovnávaní sekvencií však často používame ešte nižší prah, teda uvažujeme len zarovnania s P-hodnotou menšou ako napr. 10^{-5}. Pri malých hodnotách sú P-hodnota a E-hodnota približne rovnaké, teda taký istý prah môžme použiť aj na E-hodnotu.

2 Súvisiaca práca

V tejto kapitole si uvedieme stručný prehľad modelov, ktoré zahŕňajú doplnkové informácie do zarovnania pomocou metód klasifikácie a stručne uvedieme v čom sa bude náš model líšiť.

V princípe môžme rozlišovať dva typy modelov - *generatívny model* a *diskriminačný model*.

Konvenčné techniky odhadu pre zarovnávania sa zakladajú na generatívnom modeli. Generatívny model (napr. HMM) sa snaží modelovať proces, ktorý generuje dáta ako pravdepodobnosť $P(X, Y, Z)$, kde $X = x_1x_2 \dots x_n$, $Y = y_1y_2 \dots y_m$ a Z je zarovnanie. Ak poznáme $P(X, Y, Z)$ (alebo jej dobrý odhad),

$$\arg\max_z P(X = x, Y = y, Z = z)$$

predikuje zarovnanie z z dvoch sekvencií x a y. Aby sme zľahčili odhad $P(X, Y, Z)$, rozložíme ju pomocou nezávislých predpokladov na procese, ktorý generuje x a y. To síce vedie k efektívnym a jednoduchým problémom odhadu, ale obmedzuje to interakcie v rámci sekvencií, ktoré by sme mohli modelovať. [YJEP07]

Výskum v oblasti strojového učenia dokázal, že diskriminačné učenie (SVM, RandomForest) zvyčajne produkuje oveľa presnejšie pravidlá ako generatívne učenie(HMM, naive Bayes classifier). [YJEP07] Môže to byť vysvetlené tým, že $P(Z|X, Y)$, je už vhodné na vyhodnotenie optimálnej predikcie

$$\arg\max_z P(Z = z|X = x, Y = y).$$

[YJEP07]
Diskriminačné učenie aplikované na problém zarovnania bude priamo odhadovať $P(Z|X, Y)$ alebo prislúchajúcu diskriminačnú funkciu, a preto sa zamerá na podstatnú časť problému odhadu. [YJEP07]

Aktuálne existuje len niekoľko prístupov k diskriminačnému učeniu modelov zarovnávaní. Jeden z možných prístupov je riešiť *problém inverzného zarovnania* pomocou strojového učenia. [YJEP07]

Definícia 2.0.1 (Inverzné zarovnanie)**.** Máme dané sekvencie a k nim zarovnanie. Inverzné zarovnanie nám vráti váhový model, s ktorým daný algoritmus na zarovnávanie vráti požadované zarovnanie k daným sekvenciám.

Problém inverzného zarovnania bol prvý krát formulovaný v [GS96]. Na tomto probléme je postavený aj model v [YJEP07], kde sa na trénovanie Support Vector Machine (SVM) dá pozerať ako na riešenie tohto problému. V článku sa zaoberajú použitím *Structural SVM* algoritmu na zarovnávanie proteínových sekvencií. Diskriminačné učenie umožňuje zahrnutie množstva dodatočnej informácie – státisíce parametrov. Navyše SVM umožňuje trénovanie pomocou rôznych účelových funkcií (loss functions). SVM algoritmus má lepšiu úspešnosť ako generatívna metóda SSALN, ktorá je veľmi presným generatívnym modelom zarovnaní, ktorá zahŕňa informáciu o štruktúre.

Podobný prístup je aj v CONTRAlign [DGB06], kde sa používajú Conditional Random Fields (CRF). Tento prístup tiež ťaží z benefitov diskriminačného učenia, avšak narozdiel od [YJEP07] neumožňuje použitie účelových funkcií.

3 Klasifikácia na základe lokálnej informácie

Cieľom našej práce bolo zahrnúť dodatočnú informáciu do zarovnávania sekvencií, ktorá je poskytnutá formou anotácií k príslušným bázam. Keďže náš zarovnávač je založený na pHMM (sekcia 1.6.2), potrebujeme nejak určiť emisné pravdepodobnosti v jednotlivých stavoch. Keďže chceme okrem jednotlivých báz zahrnúť aj iné informácie, rozhodli sme sa na tento účel využiť klasifikátory. Klasifikátory určujú, ktoré pozície sa majú zarovnať k sebe a túto informáciu zakomponovávame do výsledného zarovnávača pomocou modelov, ktoré si popíšeme v kapitole 4.

V tejto kapitole sa budeme venovať výberu klasifikátora a jeho vstupných atribútov. Najskôr si stručne popíšeme algoritmus klasifikátora, potom typy vstupných atribútov, ktoré sme navrhli a nakoniec sa budeme venovať vlastnostiam klasifikátora s danými typmi atribútov a výberu finálneho klasifikátora pre použitie v našich modeloch.

3.1 Náhodné lesy

Ako klasifikátor sme si vybrali *náhodný les (angl. Random forest)*, pretože patrí v súčasnosti medzi najlepšie klasifikátory. V tejto sekcii si popíšeme základný algoritmus a v čom spčívajú jeho vyhody a jeho najdôležitejšie vlastnosti.

Dôležité vlastnosti náhodných lesov

Klasifikátor náhodný les má niekoľko dôležitých vlastností, kvôli ktorým sme si ho vybrali:

- má vysokú presnosť (podľa [BC] je najpresnejší spomedzi všetkých vtedajších klasifikačných algoritmov)

- je efektívny aj na veľkých dátach
- dokáže obsiahnuť tisícky vstupných premenných
- natrénovaný náhoný les môžme uložiť a neskôr použiť na ďalšie dáta
- dáva odhad, ktoré premenné sú dôležité na klasifikáciu.

Navyše má ešte niekoľko ďaľších užitočných vlastností, ktoré však pre nás neboli rozhodujúce:

- dokáže sa vysporiadať aj s väčším množstvom chýbajúcich dát
- produkuje interný nevychýlený odhad chyby aj počas trénovania lesa
- dokáže počítať vzťahy medzi atribútmi a klasifikáciou
- počíta vzdialenosť medzi jednotlivími pármi prípadov, čo môže byť použité pri clusteringu a detekcii outlierov alebo ponúknuť zaujímavé pohľady na dáta.
- počítanie vzdialenosti môže byť použité aj na neoanotované[1] dáta, čo sa dá použiť na klastrovanie, detekciu outlierov.
- Ponúka experimentálnu metódu na detekciu interakcie medzi premennými.

3.1.1 Klasifikačné stromy

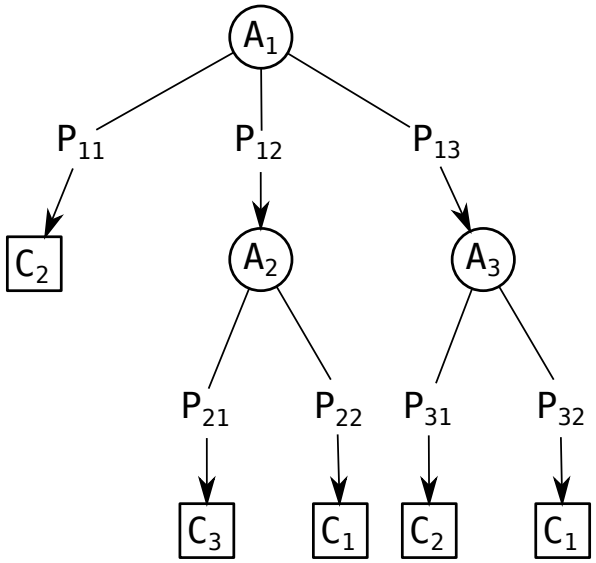
Keďže náhodný les je vlastne les *klasifikačných stromov* [2], je nevyhnutné, aby sme si najskôr povedali niečo o nich.

Klasifikačný strom je strom vybudovaný na základe trénovacej množiny, ktorý na základe vstupných údajov (vstupného vektora) predpovedá hodnotu výstupnej premennej. Klasifikačný strom sa skladá z *uzlov* a *listov*. V uzle sa nachádza rozdeľovacie kritérium, podľa ktorého sa vieme rozhodnúť do ktorej vetvy máme pokračovať. V listoch sa nachádzajú triedy, do ktorých sa vstupný vektor klasifikuje.

Klasifikácia vstupného vektora vyzerá nasledovne: prechádzame strom zhora nadol a postupne sa zaraďujeme do vetiev podľa rozdelovacích kritérií. Podľa toho, v ktorom liste skončíme, sa určí výstupná hodnota.

[1] bez informácie o triede

[2] niekedy označované ako rozhodovacie stromy



Obr. 3.1: Klasifikačný strom. A sú atribúty, P sú konkrétne hodnoty atribútov a C sú triedy, do ktorých klasifikujeme dáta.

Trénovanie

Na trénovanie rozdovacích stromov sa používajú rôzne algoritmy. My si spomenieme najjednoduchší z nich – algoritmus ID3 [Wik14b]. Budeme rozlišovať dva typy listov: uzavretý a neuzavretý list. Uzavreté listy majú už priradenú triedu. Neuzavreté listy ešte nemajú priradenú triedu. Ku každému neuzavretému list prislúcha nejaká sada príkladov.

Algoritmus začína so stromom, ktorý má práve jeden vrchol – neuzavretý list. Kým existuje nejaký neuzavretý list l opakuje:

- Ak sú v jednom liste všetky príklady jednej triedy, označí list touto triedou a získame uzavretý list.
- Inak vyberie najinformatívnejší atribút A_l a vrcholu l pridá deti, rozdelené podľa hodnôt atribútu A_l.

Najinformatívnejší atribút vyberáme pomocou miery *zisk informácie (angl. information gain)*. Zisk informácie je založený na rozdieli entrópie pred a po rozdelení podľa daného atribútu. Nech C je množina tried a p_c je pravdepodobnosť, že náhodný prvok

z S patrí do C, potom entropiu množiny S počítame ako:

$$E(S) = -\sum_{c \in C} p_c \log p_c$$

Zisk informácie potom vypočítame takto:

$$IG(S, A) = E(S) - \sum_{v \in A} \frac{\left|S_{\{A=v\}}\right|}{|S|} E\left(S_{\{A=v\}}\right)$$

3.1.2 Náhodný les

Klasifikácia

Náhodný les sa skladá z mnohých klasifikačných stromov, ktoré tvoria komisiu. Pri klasifikácii nejakého vstupného vektora sa predloží vstupný vektor každému stromu. Každý strom následne vráti klasifikáciu a hovoríme, že stromy *hlasujú*. Náhodný les následne zvolí klasifikáciu podľa väčšiny z hlasov všetkých stromov v lese.

Trénovanie

Pri trénovaní náhodných lesov sa využívajú dve hlavné techniky. Prvou z nich je takzvaný *bagging*, alebo tiež *bootstrapping*. Máme N trénovacích vektorov. Trénovaciu množinu pripravíme pre každý strom zvlášť tak, že z pôvodných N vektorov vyberieme náhodne s opakovaním N vektorov, ktoré budú použité na natrénovanie daného rozhodovacieho stromu.

Druhá technika spočiva v zmene trénovacieho algoritmu pre stromy. Nech trénovacie vektory majú M atribútov. Namiesto všetkých M atribútov sa v každom vrchole vyberie náhodne len $m << M$ z nich. Najlepší atribút na rozdelenie vrcholu sa potom vyberá z týchto m atribútov, pričom m je konštantné počas celého algoritmu.

Každý strom je potom plne natrenovaný, bez orezávania.

Parameter m je jediným parametrom, na ktorý je náhodný les citlivý. Ako bolo ukázané v [Bre01], chyba náhodného lesu závisí na dvoch veciach: korelácii medzi dvoma storomami v lese (zvýšenie korelácie vedie k zväčšeniu chyby) a sily[3] každého stromu v lese (zvýšenie sily jednotlivých stromov vedie k zníženiu chyby). Ak znížime m znížime oboje – koreláciu aj silu stromov. Optimálny rozsah m je zvyčajne celkom veľký a

[3]silný klasifikátor je taký, ktorý má malú chybu

v praxi sa zvykne často používať $m = \sqrt{M}$ alebo sa zvykne m zvoliť tak aby sa minimalizovala *out of bag*[4] chyba.

3.2 Použitie náhodných lesov na klasifikáciu zarovnaní

Keďže v našich modeloch máme dva typy stavov, použili sme dva typy klasifikátorov: *Match klasifikátor* pre Match stav a *Indel klasifikátor* pre Inzert stavy.

Klasifikátory dostanú na vstupe dáta asociované s pozíciami v daných sekvenciách a rozdeľujú ich do dvoch tried – nula a jedna. V Match klasifikátore trieda jedna znamená, že dané dve pozície majú byť zarovnané k sebe a nula znamená, že nie. V Indel klasifikátore trieda jedna označuje pozície, ktoré majú byť zarovnané k medzere a nula tie, ktoré nemajú. Ako výstupnú hodnotu pre naše modely berieme pravdepodobnosť, že dané dáta patria do triedy jedna.

Tieto pravdepodobnosti sú akousi mierou istoty daného klasifikátora, a keďže tieto dva klasifikátory sú nezávislé, súčet ich výstupov nemusí byť jedna. Sú totiž tri možnosti, čo sa môže stať:

- dve bázy sú zarovnané k sebe
- báza je zarovnaná s nejakou inou bázou (túto možnosť klasifikátory nepokrývajú)
- báza je zarovnaná s medzerou

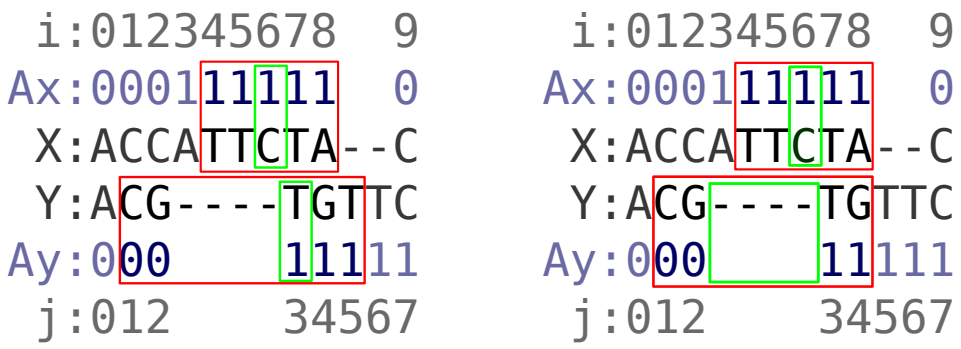
3.2.1 Výber atribútov

Klasifikátor dokáže pracovať len z poľom číselných atribútov, preto bolo treba najskôr dáta dať do tejto podoby. Navyše bolo treba rozhodnúť, aké dáta klasifikátoru podstrčíme. Potrebovali sme, aby klasifikátor videl bázy a ich anotácie na daných pozíciách. Navyše sme sa rozhodli pridať ako pomôcku aj okolie daných báz spolu s anotáciami. Toto okolie budeme volať *okno*. Vyskúšali sme štyri rôzne formy okna a zisťovali sme, ktorá forma je pre klasifikátor najvhodnejšia. Na to sme mali dve miery: úspešnosť klasifikátora a dôležitosť atribútov. Pri úspešnosti sme sledovali rozdelenie výstupu klasifikátora pre pozitívne a negatívne atribúty. Dôležitosť atribútov sme mali hlavne na kontrolu a vizualizáciu ktoré dáta považuje klasifikátor za dôležité.

[4]dáta, ktoré sa nenachádzajú v jednotlivých trénovacích množinách pre dané stromy, viac v [BC]

Definícia okna

Okno veľkosti w pozostáva z $2w$ blokov veľkosti $k = (1 + \#\text{anotácií})$. Majme teda dve sekvencie, $X = x_1x_2 \dots x_n$ a $Y = y_1y_2 \dots y_n$ a pozície i a j. Pri Match klasifikátore okno veľkosti w obsahuje $x_{i-w/2} \dots x_i \dots x_{i+(1+w)/2}$, $y_{j-w/2} \dots y_j \dots y_{j+(1+w)/2}$ a všetky anotácie príslušných báz. (Obr. 3.2a) Pri Indel klasifikátore používame tiež dve pozície – prvá je pozícia bázy, na ktorú sa pýtame a druhá je pozícia medzery medzi dvoma bázami. Predpokladajme teraz, že X je sekvencia s bázou. Okno Indel klasifikátora veľkosti w obsahuje $x_{i-w/2} \dots x_i \dots x_{i+(1+w)/2}$, $y_{j-w/2} \dots y_j \dots y_{j+(1+w)/2-1}$ a všetky anotácie príslušných báz. (Obr. 3.2b)



(a) Match klasifikátor (b) InDel klasifikátor

Obr. 3.2: Okno klasifikátora pre pozície $i = 6$ a $j = 3$

Typ dát A - okno bez úpravy

Ako prvý typ dát sme zobrali okno tak, ako sme ho definovali v predošlej sekcii. Dáta obsahujú priamo všetky bázy a anotácie tak, ako sú v okne.

Typ dát B - zhody v stĺpcoch okna

Druhý typ dát obsahuje aktuálnu bázu spolu s jej anotáciami a navyše pole veľkosti $k * w$, ktoré má na i-tom mieste jedna ak $okno_X[i] = okno_Y[i]$, ináč nula (w je veľkosť okna, k je veľkosť bloku, $okno_X$ je časť okna zodpovedajúca sekvencii X a $okno_Y$ sekvencii Y). V Indel klasifikátore je jedna malá zmena: pozícia 0 aj 1 v x-ovej sekvencii sú porovnávané s pozíciou 1 v y-ovej a pozícia 2 v x-ovej sa porovnáva s pozíciou 2 v y-

ovej. Pozíciu 1 v y-ovej sekvencii sme zopakovali pre to, že sme experimentom zistili, že pre klasifikátor je dôležitá.

Typ dát C - matica zhôd v okne

Tretí typ dát je podobný ako typ B. Rozdiel je v tom, že teraz pole obsahuje nie len zhody po dvojiciach ale celú maticu zhôd. Teda opäť máme aktuálne bázy s anotáciami a pole má veľkosť $k * w^2$. Každý riadok sa skladá z jednotlivých blokov a v tabuľke v x-tom riadku, y-tom stĺpci a i-tom mieste v bloku je jedna práve vtedy, keď $okno_X[x+i] = okno_Y[y+i]$.

Typ dát D - kombinácia A a B

Posledný typ dát je kombináciou typov dát A a B. Dáta opäť obsahujú všetky bázy a anotácie tak ako v type A a navyše sme pridali pole zhôd z dát typu B. Táto informácia je síce redundantná a klasifikátor by si ju mal vedieť odvodiť aj sám, no experimenty ukázali, že to pomôže.

3.2.2 Trénovanie a testovanie klasifikátorov

Pre oba typy klasifikátorov sme sa snažili nájsť vhodné pozitívne a negatívne trénovacie príklady. Z oboch sme do trénovacej množiny zahrnuli rovnako, aby bola trénovacia množina vyvážená. Ak by sme nemali vyváženú trénovaciu množinu, klasifikátory by zvýhodnili triedu, ktorej príkladov by bolo viacej. Napríklad, keby sme pri Indel klasifikátore zobrali všetky pozície s medzerou ako pozitívne a všetky zarovnané pozície ako negatívne príklady, klasifikátor, ktorý by dával hodnotu nula, by mal pomerne vysokú úspešnosť, pretože zarovnaných pozícií je rádovo viac ako pozícií s medzerou. Preto by natrénovaný klasifikátor mal tendenciu dávať nižšie hodnoty, aby znížil trénovaciu chybu a tomu sa chceme vyhnúť.

Výber pozitívnych príkladov bol v oboch prípadoch intuitívny. Pri výbere negatívnych príkladoch sme sa museli zamyslieť nad vhodným protipólom k pozitívnym dátam.

Výber pozitívnych a negatívnych príkladov pre Match klasifikátor

Match klasifikátor sme chceli natrénovať tak, aby pre pozície, ktoré majú byť pri sebe, dával hodnoty blízke jednej a pre pozície ktoré k sebe byť zarovnané nemajú dával hodnoty blízke nule. Ako pozitívne príklady sme teda vybrali pozície z trénovacích sekvencií, ktoré boli zarovnané k sebe.

Ako negatívne príklady sme vyskúšali dva prístupy: *náhodné dáta* a *dáta s posunom.*

Náhodné dáta: Náhodné dáta sme vyberali ako náhodné pozície, ktoré neboli zarovnané k sebe. Toto sa však ukázalo ako nedostatočné riešenie. Preto sme sa rozhodli pristúpiť k inému spôsobu výberu negatívnych vzoriek.

Dáta s posunom: Dáta s posunom sme vyberali posunutím jednej z dvoch pozícií v zarovnanom okne. Počas zarovnávania sa totiž väčšinou lokálne rozhodujeme, či zarovnať dané pozície k sebe, alebo vložiť medzeru. Ak by sme vložili medzeru, nastal by posun v jednej zo sekvencií. Rozhodli sme sa teda sa týmto inšpirovať a vyrábať negatívne príklady posunom zarovnaných pozícií. Pre každú zarovnanú pozíciu si najskôr rovnomerne náhodne vyberieme sekvenciu, ktorú budeme posúvať a potom z exponenciálneho rozdelenia náhodne vyberieme veľkosť posunu. Presný vzťah pre výpočet posunu Δ je

$$\Delta = (2D - 1) \cdot (1 + \lfloor S \rfloor) \qquad D \sim Alt(0.5),\ S \sim Exp(0.75),$$

kde prvý člen nám určuje smer posunu (čo je to isté ako: ktorá sekvencia sa bude posúvať) a druhý člen určuje veľkosť posunu (chceme celočíselnú hodnotu ≤ 1).

Výber pozitívnych a negatívnych príkladov pre Indel klasifikátor

Indel klasifikátor sme chceli natrénovať tak, aby pre miesta, ktoré majú byť zarovnané s medzerou, dával čo najvyššie hodnoty a pre miesta, ktoré nemajú byť zarovnané k medzere, dával čo najnižšie hodnoty. Ako pozitívne príklady sme sa teda rozhodli vyberať pozície, ktoré boli v trénovacích sekvenciách zarovnané k medzere. Ako negatívne príklady sme vybrali pozície, ktoré boli zarovnané k sebe, teda tie isté, čo sme pri trénovaní Match klasifikátora považovali za pozitívne. Akurát v tomto prípade mala jedna

zo sekvencií okno skrátené o jedna, teda akoby bola medzera pred bázou, s ktorou je aktuálne uvažovaná pozícia zarovnaná.

Parametre trénovacej a testovacej množiny

Dáta pre naše experimenty pochádzajú z nášho simulátora. Trénovacia sada sa skladá z 20 zarovnaní sekvencií, pričom každé zarovnanie má dĺžku 10000. Celá trénovacia pre Match klasifikátor sada má 329428 príkladov. Pre Indel klasifikátor má sada 34150 príkladov. Testovacia sada sa skladá z jedného zarovnania s dĺžkou 10000. Pričom pre Match klasifikátor obsahuje 16498 príkladov. Pre Indel klasifikátor obsahuje 1674 príkladov. Vo všetkých sadách je polovica príkladov pozitívnych a polovica negatívnych.

3.3 Výsledky experimentov

Použili sme klasifikátory tak, ako sme popísali v sekcii 3.2 a výsledky sa nachádzajú v tabuľke 3.1.

3.3.1 Dôležitosť atribútov

Atribúty sme označili takto: stredná pozícia je 0, pozície naľavo sú -1, -2 a pozície napravo 1 a 2, pričom v Indel klasifikátore v sekvencii s medzerou pozícia 0 chýba. Bázy v sekvencii X označujeme x_i a v Y y_i. Anotácie v sekvencii X označujeme a_i a v Y b_i. Zhody medzi bázami i, j budeme označovať $m_{i,j}$, pričom označenie zhôd v type dát B a D zjednodušíme na m_i. Podobne zhody medzi anotáciami označíme $l_{i,j}$ (resp. l_i).

V tabuľke 3.1 si môžeme všimnúť, že klasifikátory sa zamerali najmä na bázy a okrem prípadu C, anotácie skoro nebrali do úvahy. Toto správanie zodpovedá tomu, že v praxi bázy majú podstatne väčší význam pri zarovnávaní sekvencií.

V prípade dát typu A a B sa klasifikátory Match klasifikátory nezamerali na stredné bázy ako by sme to čakali, ale všetky bázy brali približne s rovnakou váhou. Indel klasifikátory na tom boli podobne, hoci v prípade dát typu B je vidieť výraznejšiu preferenciu atribútu zhody báz x_0, y_1 (l_0), avšak atribúty l_1 a l_2 klasifikátoru neprišli dôležité.

Typ dát	chyba		atribúty	
	trénovacia	testovacia	najvýznamnejšie	najbezvýznamnejšie
A	93,07%	83,57%	x_0, y_{-1}, y_1, x_*, y_*	$\{a, b\}_*$
B	84,05%	84.31%	m_{-1}, m_0, m_*	l_*
C	80,44%	79,79%	$m_{-2,-2}$, $m_{2,2}, l_{-1,-1}, l_{1,1}, m_{0,0}$	ostatné
D	93,65%	84,32%	$m_0, m_{-1}, m_1, m_2, m{-}2$	l_*, $\{a, b\}_*$

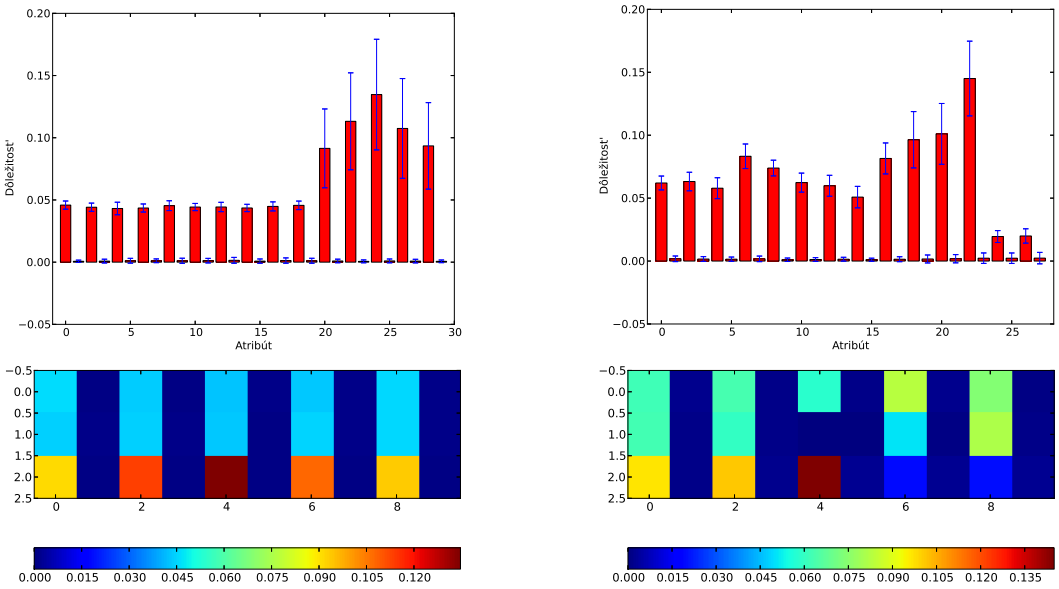
(a) Match klasifikátor

Typ dát	chyba		atribúty	
	trénovacia	testovacia	najvýznamnejšie	najbezvýznamnejšie
A	88,51%	74,13%	x_0, x_1, x_{-1}, x_*, y_*	$\{a, b\}_*$
B	77,17%	75,75%	m_0, m_{-1}, m_{-2}	l_*, m_1, m_2
C	78,03%	75,03%	$l_{2,2}$, $m_{-2,-2}$, $m_{0,1}$, $l_{-1,-1}$, x_0, a_0	ostatné
D	88,78%	76,46%	$m_0, m_{-1}, m_{-2}, x_1, \{x, y\}_*$	$\{l, a, b\}_*$

(b) Indel klasifikátor

Tabuľka 3.1: Porovnanie vlastností klasifikátorov pri rôznych typoch dát.

Pri type dát C si môžeme všimnúť, že najdôležitejšie atribúty sú na uhlopriečke, čo zodpovedá typu dát B, ibaže v tomto prípade sa nám vyskytli na niektorých pozíciách anotácie namiesto báz. Avšak ak sa nám tam vyskytla anotácia, bázu už klasifikátor nepovažoval za dôležitú a aj naopak. Indel klasifikátor navyše považoval za dôležitejšiu aj bázu na aktuálnej pozícii a jej anotáciu. Pri type dát C, na rozdiel od ostatných, klasifikátor viac berie do úvahy aj anotácie.



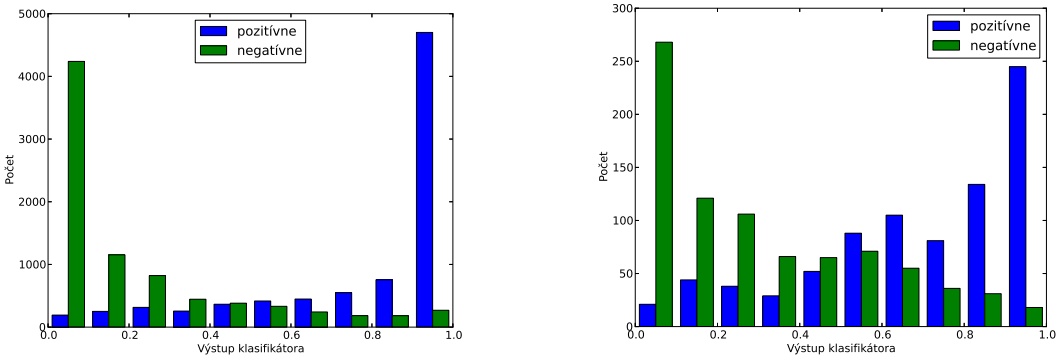
(a) Match klasifikátor (b) InDel klasifikátor

Obr. 3.3: **Dôležitosť atribútov pre typ dát D** - hodnoty sú normalizované, aby súčet bol jedna, modrý pásik označuje štandardnú odchýlku cez jednotlivé stromy v náhodnom lese. Pod grafom je tepelná mapa pre lepšiu vizualizáciu.

Pri dátach typu D sme dostali v prípade match klasifikátora očakávanú distribúciu dôležitosti atribútov (obr. 3.3). Dáta typu B boli uprednostnené voči typu A, avšak aj bázy z typu A boli dôležité. Pri Indel klasifikátore sme dostali graf, ktorý je zložením grafov z dát typu A a B. Zaujímavosťou je, že pri tomto type dát má výraznejšiu úlohu práve zhoda na stredovej pozícii, teda pozície už nie sú také rovnocenné ako to bolo v type B.

3.3.2 Úspešnosť klasifikátora

V tabuľke 3.1 vidíme, že hoci typ C obsahuje nadmnožinu atribútov B, jeho úspešnosť je nižšia a tento typ nemá zmysel používať. Ďalej si môžeme všimnúť, že kombinácia typov A a B využila klady oboch a mierne ich vylepšila.

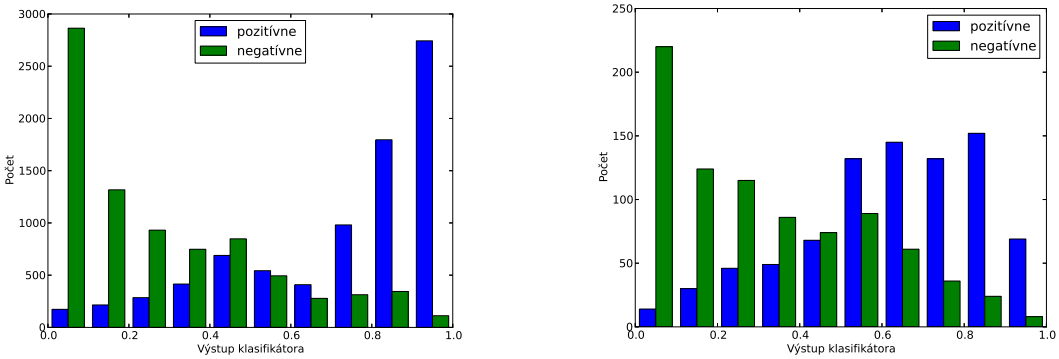


(a) Match klasifikátor

(b) Indel klasifikátor

Obr. 3.4: Distribúcia výstupu z klasifikátora pri type dát D – modré sú pozitívne príklady a zelené su negatívne. Na x-ovej osi je výstup klasifikátora a na y-je počet inštancií, pre ktoré výstup z klasifikátora padol do daného chlievika

Úspešnosť klasifikátorov s rôznym typom dát sme skúmali aj podrobnejšie, pozerali sme sa aj na distribúciu výstupov klasifikátora. Silný klasifikátor má totiž distribúciu takú, že väčšina pozitívnych príkladov má vysoké výstupné hodnoty a väčšina negatívnych klasifikátorov má nízke výstupné hodnoty. Najlepšiu distribúciu výstupov sa nám podarilo dosiahnuť práve pri type D (obr. 3.4). Na obrázku 3.5 sú pre porovnanie najhoršie distribúcie z ostatných typov dát.



(a) Typ C: Match klasifikátor

(b) Typ A: Indel klasifikátor

Obr. 3.5: Horšie distribúcie výstupov z klasifikátora – modré sú pozitívne príklady a zelené sú negatívne. Na x-ovej osi je výstup klasifikátora a na y-je počet inštancií, pre ktoré výstup z klasifikátora padol do daného chlievika

3.4 Zhrnutie

V tejto kapitole sme si predstavili klasifikátory, ktoré použijeme v našich modeloch. Uviedli sme dva typy klasifikátorov – pre Match stav a Inzert stav. Venovali sme sa výberu atribútov pre klasifikátory a ich trénovaniu. Predstavili sme si štyri množiny atribútov pre naše klasifikátory, pre ktoré sme porovnali vlastnosti klasifikátora.

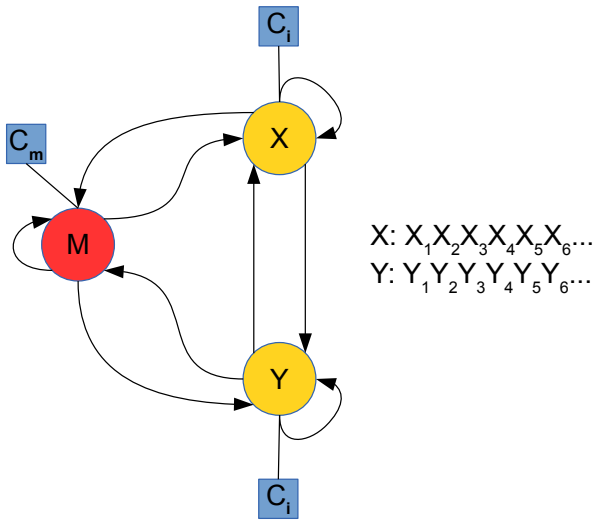
Nakoniec sme sa rozhodli použiť ako množinu atribútov typ D, pretože s týmito atribútmi mali klasifikátory najlepšie vlastnosti a aj najvyššiu úspešnosť.

4 Modely

V sekcii 1.6 sme si zadefinovali pHMM pre zarovnávanie sekvencií (obr. 1.7). V našom riešení sme predstavili 2 modifikácie pôvodného pHMM na zakomponovanie dodatočnej informácie, pričom sme využili klasifikátory. V oboch modeloch sú klasifikátory rovnaké, aj s rovnakým postupom trénovania. Líši sa len trénovanie samotného pHMM a architektúra modelu.

4.1 Model s klasifikátorom ako emisiou (Model A)

Tento model vyzerá rovnako ako základný model, aj pravdepodobnosti prechodov zostanú, ale emisnú pravdepodobnosť sme nahradili výstupom z klasifikátora.



Obr. 4.1: Model s klasifikátorom ako emisiou

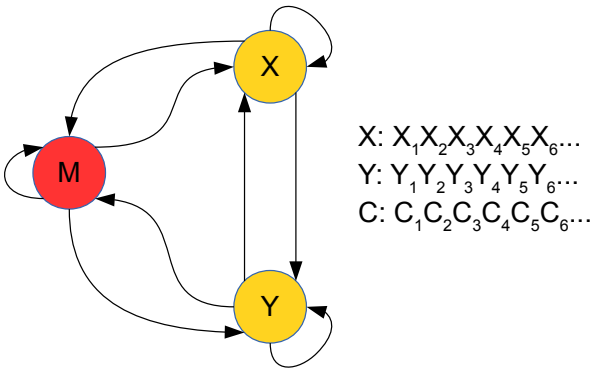
Problémom tohto modelu je, že výstup klasifikátora nezodpovedá emisným pravdepodobnostiam, ale akejsi istote klasifikátora o tom, že dve pozície majú byť zarovnané

k sebe. Hodnoty z klasifikátora teda nesumujú do 1 a model nie je korektný pravdepodobnostný model. V praxi sa však ukázalo (viď sekcia 4.4.1), že to až tak nevadí, avšak o takomto modeli už nemôžme hovoriť ako o pravdepodobnostnom. Je len inšpirovaný pHMM.

4.2 Model s klasifikátorovou páskou (Model B)

(ToDo: alebo s orákulom?)

Keďže predošlý model nie je korektný pravdepodobnostný model, navrhli sme alternatívny model, ktorý navyše modeluje aj výstup z klasifikátora. Nemodelujeme teda len dvojicu sekvencií, ale aj sekvenciu výstupov klasifikátora vo forme pásky. Tento model teda je korektný pravdepodobnostný model. Pásku s výstupom z klasifikátora považujeme za akúsi pomôcku pre náš zarovnávač.



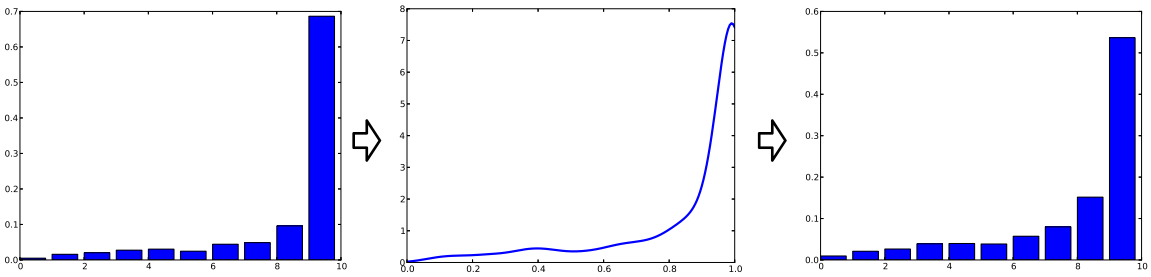
Obr. 4.2: Model s klasifikátorovou páskou

Keďže stále ide o párový HMM, pásku si musíme predstaviť ako cestu v 2D tabuľke výstupov klasifikátorov, ktorá sa zhoduje s cestou zarovnania. Teda ak sa pohneme horizontálne alebo vertikálne, používame Indel klasifikátor a ak sa pohneme diagonálne tak použijeme Match klasifikátor.

4.2.1 Diskrétna verzia

Klasifikátor môže ľubovoľnú vrátiť hodnotu z intervalu $\langle 0, 1\rangle$. Keďže výstup z klasifikátora je spojitý, museli sme ho pre naše potreby diskretizovať. Diskretizovať hodnoty

môžme 2 spôsobmi, buď priamo – spočítať histogram pre trénovacie dáta, alebo nepriamo – interpolovať vstupnú vzorku pomocou spojitej distribúcie, napr. pomocou zmesi gausiánov a potom toto rozdelenie diskretizovať (obr. 4.3). Druhá spomenutá metóda má výhodu v tom, že vyhladí šum, ale treba s ňou narábať opatrne, aby sme nezaviedli príliš veľkú nepresnosť. V oboch prípadoch si musíme zvoliť počet košov b, do ktorých dáta rozdelíme. Koše sme rozdelili rovnomerne a na základe experimentov sme si zvolili $b = 10$.



(a) Histogram (b) Zmes gausiánov (c) Nasekaný gausián

Obr. 4.3: Dve možnosti diskretizácie výstupu klasifikátora – buď použijeme histogram 4.3a, alebo dáta najskôr aproximujeme pomocou zmesi gausiánov 4.3b a následne rozsekáme na koše 4.3c.

4.2.2 Spojitá verzia

Alternatíva k predošlej metóde je použiť spojitý HMM. Hlavný rozdiel medzi spojitými a diskrétnymi HMM je, že spojité HMM nepočítajú s pravdepodobnosťou, ale s hustotou. Hodnota hustoty v danom bode narozdiel od pravdepodobnosti môže byť aj väčšia ako jedna. Jedným zo štandardných spôsobov reprezentácie hustoty v spojitých HMM je zmes gausiánov. [HHL89], takže využijeme interpoláciu takouto distribúciou a to budeme brať ako hustotu výstupu klasifikátora (obr. 4.3b).

Ako vidíme z tabuľky 4.1, spojitá verzia modelu sa nám neosvedčila. Má totiž nedostatok, že distribučná funkcia nedosahuje maximum pri výstupe klasifikátora rovnom jedna, ale kúsok predtým, čo znamená istú penalizáciu v prípade, že si je klasifikátor príliš istý. To, či použijeme vyhladenie pomocou gausiánu až tak nezaváži, ale rozhodli sme sa ho predsa len využiť, pretože úspešnosť bola trochu vyššia takmer na všetkých

testovaných sekvenciách.

	Úspešnosť
Histogram	82,25%
Nasekaný gausián	82,98%
Zmes gausiánov	65,59%

Tabuľka 4.1: Porovnanie úspešností modelu B pri rôznom spracovaní pásky.

4.3 Trénovanie modelov

4.3.1 Trénovanie modelu A

V modeli A sme trénovali iba prechodové pravdepodobnosti, emisie sme mali priamo z natrénovaného klasifikátora. Prechodové pravdepodobnosti sme trénovali pomocou metôdy maximálnej vierohodnosti, tak ako sme si to popísali v kapitole 1.6.4.

4.3.2 Trénovanie modelu B

V modeli B sme trénovali aj tranzície aj emisie, opäť metôdou maximálnej vierohodnosti. Pri trénovaní emisií sme však urobili malú zmenu. Aby sme vedeli ľahko interpolovať vstupnú vzorku a z dôvodu lepšej vizualizácie sa nám hodí modelovať pravdepodobnosti $P(C|X \cap Y)$ (resp. $P(C|X)$ a $P(C|Y)$). Ukážeme si, že pravdepodobnosť $P(X \cap Y \cap C)$ vieme rozložiť pomocou $P(C|X \cap Y)$ a $P(X \cap C)$ vieme rozložiť pomocou $P(C|X)$.

$P(X \cap Y)$ poznáme z frekvenčnej tabuľky a $P(C)$ vieme rozložiť pomocou nasledujúcej vety.

Veta 4.3.1 (Veta o úplnej pravdepodobnosti)**.** *Nech $A_1 \dots A_n$ tvoria rozklad univerza Ω a nech B je udalosť, potom*

$$P(B) = \sum_{i=1}^{n} P(B|A_i)P(A_i)$$

Máme teda

$$P\left[C = c\right] = \sum_{\forall x \in X, y \in Y} P\left[C = c | X = x \wedge Y = y\right] P\left[X = x \wedge Y = y\right],$$

pričom druhý člen poznáme a $P\left[C = c | X = x \wedge Y = y\right]$ už vieme ľahko dopočítať. Keď máme fixnuté dve bázy, vieme vybrať z trénovacej sekvencie všetky pozície, kde sú zarovnané tieto bázy a vypočítať distribúciu C.

Pre Indel stav postupujeme analogicky, ibaže namiesto $P(X \cap Y)$ máme buď $P(X)$ alebo $P(Y)$ podľa toho, ktorý stav práve počítame.

Vieme teda pravdepodobnosť emisie (x, y, c) Match stave vypočítať ako

$$P\left[C = c \wedge X = x \wedge Y = y\right] = P\left[C = c | X = x \wedge Y = y\right] P\left[X = x \wedge Y = y\right]$$

a pravdepodobnosť emisie (x, c) resp. (y, c) v Inzert stavoch pomocou

$$P\left[C = c \wedge X = x\right] = P\left[C = c | X = x\right] P\left[X = x\right]$$
$$P\left[C = c \wedge Y = y\right] = P\left[C = c | Y = y\right] P\left[Y = y\right].$$

Emisie môžme teda natrénovať zvlášť pre každú dvojicu báz (resp. pre každú bázu).

4.3.3 Popis trénovacej a testovacej množíny

V tejto sekcii sa budeme zaoberať len množinami pre trénovanie parametrov modelu. Množiny pre trénovanie klasifikátora sme si popísali v kapitole 3.2.2.

V prípade simulovaných dát 1 a 2 boli trénovacie množiny pre naše modely jedno zarovnanie dĺžky 10000. Testovacie množiny boli 6 zarovnaní dĺžky 1000. Zarovnania boli vygenerované našim simulátorom. V tabuľke 4.2 uvádzame pravdepodobnosť mutácie pre jednotlivé dáta.

4.4 Výsledky experimentov

4.4.1 Porovnanie modelov

Hlavný rozdiel v modeloch A a B, ktoré sme predstavili v sekciách 4.1 a 4.2 je v tom, že prvý model je diskriminačný, zatiaľčo druhý je generatívny. (ToDo: mozno toto lepsie sformulovat alebo vyhodit lebo si nie som isty ci je to celkom pravda)

anotácia X	anotácia Y	sim1	sim2
1	1	0,20	0,01
1	0	0,35	0,30
0	1		
0	0	0,40	0,99

Tabuľka 4.2: Pravdepodobnosť mutácie v simulovaných dátach 1 a 2

bázy	výstup klasifikátora	emis. pravd.
AA	0.96	0,09399
AG	0.96	0,00513
AA	0.42	0,00729
AA	0.26	0,00411
AG	0.02	0,00168

Tabuľka 4.3: Porovnanie emisných pravdepodobností pre rôzne bázy a výstup natrénovaného klasifikátora

Model A emituje len na základe natrénovaného klasifikátora, takže čokoľvek dokážeme klasifikátor naučiť, môžme priamo použiť. Model B používa klasifikátor iba ako pomôcku. Model funguje podobne ako štandardný model pre zarovnanie (sekcia 1.6.2) a klasifikátor iba mierne upravuje výsledné pravdepodobnosti.

Ako môžme v tabuľke 4.3 vidieť, pre rovnaký výstup klasifikátora, ale rôzne bázy sa emisná pravdepodobnosť môže líšiť. Dokonca aj pre značne nižší výstup klasifikátora pri rovnakých bázach môže byť pravdepodobnosť emisie stále vyššia ako pri rôznych bázach s vysokým výstupom z klasifikátora. Na druhej strane si môžme tiež všimnúť, že pri nižšom výstupe klasifikátora emisná pravdepodobnosť rapídne klesá a pri dostatočne nízkom výstupe klasifikátora pri rovnakých bázach už je emisná pravdepodobnosť nižšia, ako pri rôznych bázach s vysokým výstupom klasifikátora.

V tabuľke 4.4 sme porovnávali, ako sa modely dokážu prispôsobiť zložitejším dátam. Môžme si všimnúť, že model A sa správa konzistentne na oboch typoch dát, zatiaľ čo model B sa pri zložitejšom type dát nechal stiahnuť dole nutnosťou dodržovať pravde-

ToDo: pregenerovať s oknom 9

	Model A	Model B	Ref. model
Simulované 1	79,75%	84,35%	85,78%
Simulované 2	72,62%	18,57%	6,78%

Tabuľka 4.4: Porovnanie úspešností modelov. Referenčný model je obyčajný pHMM na zarovnávanie DNA sekvencií. Úspešnosť je počítaná ako percentuálna zhoda originálneho a nového zarovnania. Simulované dáta 1 sa snažia napodobňovať biologické procesy, Simulované dáta 2 nezodpovedajú biologickým dátam a sú zložitejšie na natrénovanie pre referenčný model.

podobnosti generovania báz.

4.4.2 Rôzne veľkosti okna

V tejto sekcii sme sa venovali hľadaniu optimálnej veľkosti okna. Vyskúšali sme viacero možností a výsledky sme zapísali do tabuľke 4.8.

Veľkosť okna	Model A	Model B
1	76,95%	57,86%
3	70,62%	81,42%
5	73,95%	82,98%
7	78,31%	83,05%
9	79,75%	84,35%
11	79,28%	83,57%
13	80,58%	83,98%
15	81,62%	83,98%

Tabuľka 4.5: Porovnanie úspešností modelov pri rôznej veľkosti okna.

Z tabuľky vidíme, že pre model B je optimálna veľkosť okna 7-9, pri väčších oknách už úspešnosť nerastie. Pri modeli A rástla úspešnosť až po veľkosť okna 15. Optimálnu hodnotu pre model A sme na základe tabuľky určili na 9-15.

4.4.3 Vplyv dodatočnej informácie na zarovnanie

V tejto sekcii sme zisťovali, aký vplyv má dodatočná informácia na zarovnanie. Vyskúšali sme zakázať anotácie pri veľkosti okna jedna a 9. Pri veľkosti okna rovnej jedna nemáme okrem anotácie žiadnu ďalšiu informáciu. Pri veľkosti okna 9 máme aj okolie sekvencie.

Veľkosť okna	Anotácia	Model A	Model B
1	nie	74,42%	18,35%
1	áno	76,95%	57,86%
9	nie	78,43%	82,67%
9	áno	79,75%	84,35%

Tabuľka 4.6: Vplyv dodatočnej informácie na zarovnanie.

V tabuľke 4.6 si môžme všimnúť, že zatiaľ čo pri modeli A je úspešnosť veľmi podobná, hoci aj anotácie aj väčšie okno mierne pomohli. Naopak pri modeli B sú rozdiely výrazne väčšie, vidíme, že aj samotná anotácia výrazne pomohla a väčšie okno je pre tento model nutnosťou.

4.4.4 Použitie jedného klasifikátora pre Match aj Inzert stav

V tejto sekcii sme vyskúšali nahradenie Indel klasifikátora Match klasifikátorom, ktorý vracia opačné hodnoty. Teda trénujeme len jeden klasifikátor.

	Model A	Model B
1 klasifikátor	74,53%	82,72%
2 klasifikátory	79,75%	84,35%

Tabuľka 4.7: Porovnanie použitia jedného alebo dvoch typov klasifikátorov.

Z tabuľky 4.7 vidno, že použitie dvoch typov klasifikátorov má svoje opodstatnenie, hoci naše modeli dokážu fungovať aj z jedným.

4.4.5 Porovnanie modelov s existujúcimi zarovnávačmi

V tejto sekcii sme porovnali naše výsledky s výsledkami referenčného modelu a zarovnávača *muscle* [Edg04].

Dáta	Model A		Model B		Ref. Model		Muscle	
	Zhoda	Tranz.	Zhoda	Tranz.	Zhoda	Tranz.	Zhoda	Tranz.
Simulované 1	79,75%	44,97%	84,35%	56,5%	85,78%	61,03%		
Simulované 2								
Biologické								

Tabuľka 4.8: Porovnanie našich modelov s referenčným modelom a zarovnávačom muscle.

4.5 Zhrnutie

5 Implementácia

ToDo: daky pokec o tom ze som to pisal v pythone a preco je python super

5.1 Použité knižnice

5.1.1 Realigner

ToDo: cosi k micovmu kodu - mozno referencia na jeho dizertacku + mozno nejaky pokec k tomu ze co sme pouzili a mozno ako to funguje

5.1.2 Pythonové knižnice

ToDo: python kniznice - najma scikit-learn, numpy, track

5.2 Triedy na zarovnávanie s klasifikátorom

5.2.1 Predspracovanie dát

ToDo: DataPreparers, class diagram

5.2.2 Zovšeobecnenie klasifikátora

ToDo: PairClassifier

5.2.3 HMM stavy s klasifikátorom

ToDo: Classifier state...

5.3 Pomocné programy

5.3.1 Simulátor

Simulátor sĺúži na overenie funkčnosti zarovnávača. Náhodne vygeneruje 2 sekvencie, ktoré vznikli zo spoločného predka a vyrobí korektné zarovnanie. Okrem toho vyrobí aj nejaké dodatočné informácie ktoré majú pomôcť pri zarovnávaní.

Algoritmus

Simulátor vygeneruje informáciu o tom, ktoré časti sekvencie prislúchajú génom a ktoré nie. Informácia má podobu boolovského vektora. Simulátor najskôr vygeneruje *základnú (master) postupnosť* a z nej odvodí dve ďalšie postupnosti, ktoré zodpovedajú sekvenciám.

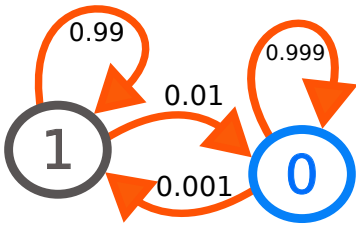
Okrem toho simulátor vygeneruje dve sekvencie, pričom prvú vyrobí náhodne a druhú odvodí z nej pomocou mutácie a inzercie/delécie. V našom prípade sme inzerciu vynechali a simulujeme ju ako deléciu v druhej sekvencii. Pri odvodzovaní bude používať aj informáciu o tom, ktorá časť je gén a ktorá nie.

Keďže simulátor vie spôsob akým generoval druhú sekvenciu z prvej, vie aj korektné zarovnanie.

Simulátor má vopred daných niekoľko konštánt – pravdepodobnosti udalostí, ktoré môžu nastať.

Generovanie informácie o génoch

Ak sa na danom mieste nachádza gén, označíme to 1 inak 0. Gény bývajú súvislé úseky, takže ich treba generovať tak, že niekedy začneme gén, potom generujeme 1, potom skončíme gén a generujeme 0. Potom môžme opäť začať gén atď.



Obr. 5.1: Markovova reťaz použitá na generovanie informácie o génoch

Generovanie robíme pomocou *Markovovskej reťaze (Markov Chain)* obr. 5.1. Generujeme podľa aktuálneho stavu a v každom kroku sa podľa pravdepodobnosti rozhodneme či sa prepneme do iného stavu alebo ostaneme v tom istom. Rozhodnutie robíme pomocou *falošnej mince (biased coin)*, kde hlava padne s určitou pravdepodobnosťou, ktorú vopred nastavíme.

Máme vygenerovanú master postupnosť a z nej teraz vyrobíme dve postupnosti pre sekvencie tak, že skopírujeme master sekvenciu, pričom každú 1 s určitou pravdepodobnosťou (v našom prípade $0,1$) zmeníme na 0.

Simulácia mutácie

Máme vygenerovanú sekvenciu a ideme vyrobiť zmutovanú sekvenciu. Spravíme to tak, že s určitou pravdepodobnosťou sa nahradí báza z pôvodnej sekvencie inou bázou. Pravdepodobnosť závisí aj od toho, či je na danej pozícii gén v oboch sekvenciách, v jednej, alebo v žiadnej. Na rozhodnutie používame jednu z troch falošných mincí podľa toho, ktorá z možností nastala (gén v oboch sekvenciách, v jednej, alebo žiadnej).

Simulácia delécie

Deléciu simulujeme opäť pomocou Markovovskej reťaze, pretože pri evolúcii majú tendenciu vypadávať súvislé úseky. Pravdepodobnosť, že začneme mazať je $0,01$ a že prestaneme $0,1$. Ak mažeme, nahradzujeme danú bázu znakom `'-'`.

Využitie

Simulátor je prvá vec, ktorú sme implementovali a slúžil hlavne na prvotné experimenty.

ToDo: vieme ho prisposobovat a merat na nom korektnost a uzitocnost modelu, alebo nieco na ten styl

5.3.2 Trénovanie modelov

ToDo: modeltraining.py

5.3.3 Testovanie klasifikátora

ToDo: random_forest_evaluation.py

5.4 Použitie

ToDo: v kratkosti o tom ako to vobec spustit so svojimi sekvenciami a modelmi...

ToDo: mozno sem dat aj moznosti rozsirenia

ToDo: konfiguracia - constants.py a config.py

Záver

Literatúra

[BC] Leo Breiman and Adele Cutler. Random forests. [Online; accessed 14-Jan-2013].

[BPSS11] Alvis Brazma, Helen Parkinson, Thomas Schlitt, and Mohammadreza Shojatalab. EBI Research - Functional Genomics - Introduction To Biology. 2011. [Online; accessed 26-Oct-2012].

[Bre01] L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[BV11] Broňa Brejová and Tomáš Vinař. *Metódy v bioinformatike [Methods in Bioinformatics]*. Knižničné a edičné centrum FMFI UK, 2011. Lecture notes.

[DEKM98] R. Durbin, S.R. Eddy, A. Krogh, and G. Mitchison. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, 1998.

[DGB06] Chuong Do, Samuel Gross, and Serafim Batzoglou. Contralign: Discriminative training for protein sequence alignment. In Alberto Apostolico, Concettina Guerra, Sorin Istrail, Pavel Pevzner, and Michael Waterman, editors, *Research in Computational Molecular Biology*, volume 3909 of *Lecture Notes in Computer Science*, pages 160–174. Springer Berlin / Heidelberg, 2006. 10.1007/11732990_15.

[Edg04] Robert C Edgar. Muscle: multiple sequence alignment with high accuracy and high throughput. *Nucleic acids research*, 32(5):1792–1797, 2004.

[GS96] D. Gusfield and P. Stelling. [28] parametric and inverse-parametric sequence alignment with xparal. *Methods in enzymology*, 266:481–494, 1996.

[HHL89] XD Huang, Hsiao-Wuen Hon, and Kai-Fu Lee. Multiple codebook semi-continuous hidden markov models for speaker-independent continuous speech recognition. 1989.

[Hir75] D. S. Hirschberg. A linear space algorithm for computing maximal common subsequences. *Commun. ACM*, 18(6):341–343, June 1975.

[KA90] S Karlin and S F Altschul. Methods for assessing the statistical significance of molecular sequence features by using general scoring schemes. *Proceedings of the National Academy of Sciences*, 87(6):2264–2268, 1990.

[MB06] Alexander Yu. Mitrophanov and Mark Borodovsky. Statistical significance in biological sequence analysis. *Briefings in Bioinformatics*, pages 2–24, 2006.

[NJ02] Andrew Y Ng and Michael I Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. *Advances in neural information processing systems*, 2:841–848, 2002.

[NW70] Saul B. Needleman and Christian D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443 – 453, 1970.

[Sri] Sargur N. Srihari. Machine Learning: Generative and Discriminative Models. `http://www.cedar.buffalo.edu/~srihari/CSE574/Discriminative-Generative.pdf`. [Online; accessed 14-Jan-2013].

[Sut07] Ivan Sutóris. Tvorba klasifikačných stromov pri procese data miningu, 2007.

[SW81] T.F. Smith and M.S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195 – 197, 1981.

[Wik14a] Wikipedia. Maximum likelihood. `http://en.wikipedia.org/wiki/Maximum_likelihood`, 2014. [Online; accessed 17-Apr-2014].

[Wik14b] Wikipedia. Maximum likelihood. `http://en.wikipedia.org/wiki/ID3_algorithm`, 2014. [Online; accessed 26-Apr-2014].

[Wik14c] Wikipedia. Maximum likelihood. `http://en.wikipedia.org/wiki/Random_forest`, 2014. [Online; accessed 26-Apr-2014].

[YJEP07] Chun-Nam Yu, Thorsten Joachims, Ron Elber, and Jaroslaw Pillardy. Support vector training of protein alignment models. In Terry Speed and Haiyan Huang, editors, *Research in Computational Molecular Biology*, volume 4453 of *Lecture Notes in Computer Science*, pages 253–267. Springer Berlin / Heidelberg, 2007. 10.1007/978-3-540-71681-5_18.