TARTU ÜLIKOOL
MATEMAATIKA-INFORMAATIKA TEADUSKOND
Arvutiteaduse instituut
Tarkvarasüsteemide õppetool
Informaatika eriala

Triinu Tasa

Kaalumaatriksite kasutamine geenide regulatsiooni mõjutavate signaalide kirjeldamiseks ja otsimiseks

Bakalaureusetöö (10 AP)

Juhendaja: Jaak Vilo, PhD

Autor: …… "….." mai 2005
Juhendaja: ……. "….." mai 2005
Õppetooli juhataja: …… "….." mai 2005

Tartu 2005

Sisukord

1. Peatükk

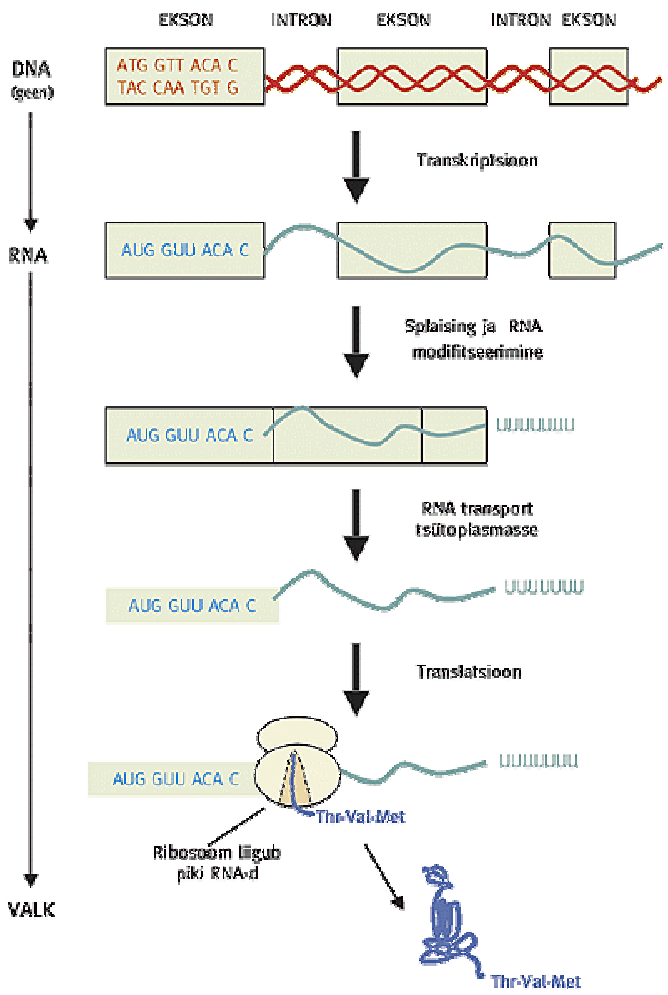
Sissejuhatus

Bioinformaatika üks suuremaid uurimisvaldkondi on *geeniregulatsioon* - uuritakse, kuidas DNA-st täpselt valku sünteesitakse. Geeniregulatsiooni kaks põhilist etappi on *transkriptsioon*, mille käigus DNA-st sünteesitakse RNA, ja *translatsioon*, ehk RNA transleerimine valguks (vt. joonis 1.1). Käesolevas töös keskendutakse geeniregulatsiooni esimesele osale ehk transkriptsioonile.

On teada, et transkriptsiooni algatavad valgud, nn. *transkriptsioonifaktorid*. Need seonduvad DNA-l kindlatele kohtadele, nn. *transkriptsioonifaktorite seondumissaitidele*, ning käivitavad sellega transkriptsiooni. Need seondumissaidid on lühikesed (4-20 nukleotiidi) alamjärjestused, mis paiknevad enamasti *promootorites* e. geenide ees asuvates piirkondades. Käesoleva töö eesmärk on koostada tööriist seondumissaitide võimalikult täpseks ennustamiseks.

Sarnase eesmärgi nimel on loodud teisigi rakendusi. Näiteks võib tuua tööriista SPEXS, mis kuulub Expression Profileri juurde [Vil02, VKK+03, url:SPEXS]. SPEXS otsib sisendjärjestustest sageli esinevaid mustreid ja väljastab need. Väljastatavad mustrid on aga sageli teineteisega väga sarnased, erinedes vaid mõnes positsioonis. Käesoleva töö eesmärk on töödelda edasi sellist väljundit.

Esimene ülesanne on SPEXS'i või mõne muu sarnase tööriista väljundi klasterdamine. See on vajalik, kui soovitakse saada kõigist väljastatavatest mustritest vaid erinevaid. Põhjuseks võib olla lihtsalt soov saada selge ülevaade olulistest signaalidest. Käesoleva töö eesmärk on aga kasutada saadud klastreid edasi uute signaalide otsimiseks.



Joonis 1.1. ***Geeniregulatsiooni põhietapid.*** *DNA informatsiooni alusel tehakse transkriptsiooni käigus RNA, mis modifitseeritakse ning viiakse tuumast tsütoplasmasse. Tsütoplasmas toimuva translatsiooni tulemusena valmistatakse RNA informatsiooni alusel aminohapete järjestusest koosnev valk [url: EGK].*

Käesoleva töö praktiline osa ongi mõeldud eelkõige täiendamaks SPEXS'i eemärgiga muuta oluliste mustrite otsimist täpsemaks, vähendada valepositiivseid ja valenegatiivseid. Praktiline töö on jaotatud kolme ossa: klasterdamine, mitmene joondamine ja maatriksite sobitamine tekstile. Klasterdamise ülesandeks on grupeerida SPEXS'i väljundis olevad mustrid, nii et jääks vaid paar üldist motiivi. Selleks, et grupeeritud mustreid üheselt kujutada, on vajalik need joondada ning joonduse põhjal saab mustrite grupist moodustada maatriksi. Selleks, et leida meid huvitavatest järjestustest moodustatud maatriksite abil uusi signaale, on vaja maatrikseid tekstile sobitada.

Autori semestritöö käigus käsitleti kahe esimese rakenduse, st. mustrite klasterdamise ja mitmese joondamise probleemi. Mõlemale probleemile pakuti välja lahendus ning koostati ka vastavad programmid. Bakalaureusetöö kontributsioon teemale seisneb kolmanda ja kõige olulisema rakenduse loomises, milleks on maatriksite sobitamine tekstile. Maatriksite sobitamise meetodite kiiruseid võrdles oma diplomitöös Marek Zäuram. Käesoleva töö käigus valminud tööriista eesmärk on rahuldada kõikvõimalikke sobitamisele seatud tingimusi – teatud arvu parimate esinemiste leidmine, teatud väärtusest paremate esinemiste leidmine jne. Selle rakenduse abil on võimalik ära kasutada klasterduse ja joondamise tulemusi uute huvitavate leidude tegemiseks. Töö loetavuse ja motiivi mõistmise seisukohast lähtudes on käesolevas töös kolmandas ja neljandas peatükis ära toodud ka mustrite klasterdamise ja mitmese joondamise probleemide kirjeldused ja meie pakutud lahendused.

Käesolev töö on jaotatud kuueks peatükiks. Teises peatükis selgitatakse edaspidi vajaminevaid mõisteid. Kolmandas antakse ülevaade klasterdamise probleemist ja meie pakutud lahendusest. Neljas peatükk käsitleb mitmese joondamise probleemi ja meie lahendust sellele ning viiendas peatükis on kirjeldatud maatriksite sobitamisega seotud probleeme. Kuuendas peatükis antakse ülevaade tulemustest.

2. Peatükk

Definitsioonid

Käesolevas peatükis defineeritakse mõisted, mida edaspidi sagedamini kasutatakse. Kuna edaspidi kasutatakse ka mõningaid statistilisi mõisteid, on ka need siin defineeritud.

2.1 Sõne

Tähistagu Σ sümbolite hulka e. *tähestikku.* Tähestiku Σ võimsust tähistatakse $|\Sigma|$. DNA puhul $\Sigma = \{A, C, G, T\}$ ning $|\Sigma|$=4. Järjestust $S = a_1 a_2 ... a_n$, kus $n \geq 0$ ja $\forall a_i \in \Sigma$ nimetatakse *sõneks* e. *järjestuseks* e. *stringiks* [Vil02]. Sõne *S* pikkus $|S|$ on *n*. Sõnet pikkusega 0 ehk tühistringi tähistatakse λ. Kõigi võimalike sõnede hulka üle tähestiku Σ tähistatakse Σ^*.

Tähemärke eristatakse sõnes teineteisest positsioonide järgi. Positsioonil *i* asuvale sümbolile a_i viitab ka tähistus $S[i]$. Tähemärgi positsioonid mittetühjas sõnes *S* asuvad vahemikus $1 \leq i \leq |S|$, st. sõne esimene sümbol asub positsioonil 1 ning viimane positsioonil $|S|$.

Sõne *S* järjestikused tähed $a_i ... a_j$ moodustavad *S* alamsõne, mis algab positsioonis *i* ning lõppeb positsioonis *j*. Tähistame seda alamsõnet $S[i..j]$, kus $1 \leq i \leq j \leq |S|$. Alamsõne alternatiivne definitsioon, mis ei kasuta tähtede positsioone, väidab et *x* on *S* alamsõne, kui $S = yxz$ suvaliste sõnede *y* ja *z* puhul.

Alamsõne $S[i..i]$ pikkus on 1 ning see vastab tähemärgile positsioonil *i*. Alamsõne $S[i..j]$ pikkus on $j - i + 1$. Me ütleme, et alamsõne $S[i..j]$ esineb sõnes *S* positsioonil *j*. Me ütleme, et alamsõne *x* esineb sõnes *S* mitu korda, kui $x = S[i..j] = S[i'..j']$ ja $j \neq j'$ [Vil02].

2.2 Muster

Mustrit kasutatakse mitmes tähenduses. Käesolevas töös pean mustri all silmas lühikest korduvat järjestust, mille struktuuri täpsema kirjelduse annan järgnevalt.

Olgu $\Sigma = \{a_1, ..., a_m\}$ *põhitähestik* ja olgu $L(a_i) = \{a_i\}$ a_i poolt defineeritud *keel*. Olgu $g_1, ..., g_n$ tähestiku Σ mittetühjad alamhulgad nii, et igas hulgas on rohkem kui üks element. Alamhulkade g_i nimetamiseks kasutame tähestikku $\Gamma = \{b_1, ..., b_n\}$, kusjuures $\Gamma \cap \Sigma = \varnothing$. Defineerime keele $L(b_i) = g_i$ ning nimetame sümboleid b_i *grupisümboliteks*.

Olgu $. \notin \Gamma \cup \Sigma$. Defineerime $L(.) = a_i$, kus $a_i \in \Sigma$. Nimetame '**.**' metasümboliks (*wildcard*) keeles Σ, $|.| = 1$.

Definitsioon. Muster on sõne üle tähestiku $\Sigma \cup \Gamma \cup \{.\}$. Mustri $\pi = c_1...c_r$, kus $c_i \in \Sigma \cup \Gamma \cup \{.\}$, keele $L(\pi)$ defineerime järgmiselt:
$L(\pi) = \{\zeta_1...\zeta_r \mid \zeta_i \in L(c_i)\}$.
Käesolevas töös kasutatakse mustrite kirjeldamisel eelpool toodud formaalse tähistusviisi asemel loetavuse parandamiseks regulaaravaldisi. Järgnev näide kirjeldab nendevahelist seotust.

Näide.

$\Sigma = \{A, C, G, T\}$
$b = \{A, C\}$
$c = \{T, G\}$
$\Gamma = \{b, c\}$
$AbcG = A[A, C][T, G]G$

2.3 Maatriks e. profiil

On teada, et sarnase funktsiooniga järjestused on ka struktuurilt sarnased. Selleks, et kirjeldada kompaktselt järjestuste struktuuri, mille bioloogiline funktsioon on sama, kasutatakse mitmeid meetodeid.

Näide: Olgu meil kolm järjestust, mis esitavad transkriptsiooni faktori kolme erinevat seondumissaiti:

GATGAG
GATGAT
TGATAT

Üks võimalus neid kõiki üheselt määratleda, on kasutada nn. *konsensust* – sõne, mille igas positsioonis on vastava positsiooni kõige populaarsem täht. Praegusel juhul on konsensuseks *GATGAT*. See aga esitab vaid ühe võimaliku näite, kuid ei kirjelda varieeruvust.

Täpsemalt saab neid kolme sõnet kirjeldada regulaaravaldise abil. Käesolevas töös kasutatakse Perli dokumentatsioonile vastavat regulaaravaldiste notatsiooni, selle põhjal vastab näitele järgmine *regulaaravaldis*: [GT][AG][TA][GT]A[GT]. Regulaaravaldis katab küll kõik võimalused, kuid annab liiga palju võimalikke variante ja ei võta arvesse tegelikke tähtede sagedusi. Selle probleemi lahendab aga *maatriksite* e. *profiilide* kasutamine.

Bioinformaatikas väljendab maatriks e. profiil nukleiinhappe või valgu segmenti suguluses olevate sekventside peres [WNB00] või matemaatilisemalt öeldes, profiil on sekventsi tõenäosuslik kirjeldus. Maatriks määrab igal positsioonil tähestiku elementide tõenäosusjaotuse [RMV03].

Maatriksite konstrueerimise viise on mitmeid. Erinevate probleemide lahendamiseks sobivad tavaliselt erinevat tüüpi maatriksid.

Kõige lihtsamat tüüpi maatriks on *joondus- e. loendusmaatriks*, kus summeeritakse kokku tähtede esinemiste arvud positsiooniti. Eelpool toodud näite põhjal koostatud joondusmaatriks näeb välja järgmine:

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

Joondusmaatriksi põhjal genereeritakse kõik teised maatriksitüübid. Otseselt joondusmaatriksist tuleneb *tõenäosusmaatriks F*, kus tähtede esinemise arvu asemel antakse tähtede esinemistõenäosused vastaval positsioonil. Eelpool toodud joondusmaatriksi tõenäosuslik kuju on järgmine:

A	0	0.7	0.3	0	1	0
C	0	0	0	0	0	0
G	0.7	0.3	0	0.7	0	0.3
T	0.3	0	0.7	0.3	0	0.7

Kuigi andmebaasides on maatriksid enamasti loendusmaatriksi või tõenäosusmaatriksi kujul, on rakendustes enim kasutatavad erinevad *kaalumaatriksid*, mida sageli nimetatakse ka *informatsiooni sisalduse maatriksiteks*. Põhjus peitub selles, et informatsiooni sisalduse maatriksi põhjal saab anda statistilise hinnangu joonduse headusele.

Valemeid kaalumaatriksite koostamiseks on mitmeid, kuid erinevused on väiksed. Üks levinumaid, mida kasutatakse ka praktilises töös, on järgmine (0.1):

$$weight_{i,j} = \ln \frac{n_{i,j} + p_i}{(N+1)\,p_i},$$

(0.1)

kus N – järjestuste arv maatriksi koostamisel (eespool toodud näites 3),

p_i – *a priori* tähe sagedus (taustasagedus),

$n_{i,j}$ - loendusmaatriksi element.

Sageli kasutatakse valemi (0.1) asemel ka lihtsustatud valemit (0.2):

$$weight_{i,j} = \ln \frac{n_{i,j} + p_i}{(N+1)\,p_i} \sim \ln \frac{f_{i,j}}{p_i},$$

(0.2)

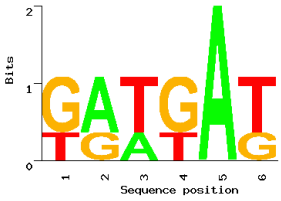
kus $f_{i,j}$ – tähe sagedus kohal *j* (tõenäosusmaatriksist).

Kaalumaatriks näeb siis eelneva näite põhjal välja järgmine:

A	–1.09	1.10	0.51	–1.09	1.47	–1.09
C	–1.09	–1.09	–1.09	–1.09	–1.09	–1.09
G	1.10	0.51	–1.09	1.10	–1.09	0.51
T	0.51	–1.09	1.10	0.51	–1.09	1.10

Positiivne kaal näitab, et tähe tõenäosus antud positsioonis on suurem kui tähe *a priori* tõenäosus [url:COL].

Käesolevas töös tuleb juttu ka maatriksite graafilisest esitusest e. *logodest*. Eelpool toodud näite põhjal moodustatud logo näeb välja järgmine:



Logo tähtede kõrguse arvutamisel kasutatakse jällegi logaritmi omadusi. Koostatakse *infosisalduse tabel A*, kasutades valemit (0.3):

$$A_{i,j} = -f_{i,j} * \log_2 f_{i,j}, \tag{0.3}$$

kus $f_{i,j}$ on tõenäosusmaatriksi vastav element.

Järgmisena leitakse iga veeru elementide summa *H*:

$$H_j = -\sum_i f_{i,j} * \log_2 f_{i,j}$$

Logo veeru kõrguseks on 2-*H* ning tähe kõrgus on suhteline osa vastavast veerust: $p_i * (2 - H)$.

Kaalumaatriksid on ühe piiranguga – nimelt eeldatakse, et tähed on positsiooniti sõltumatud, st. tähe esinemist vastavas positsioonis ei mõjuta eelnevatel positsioonidel esinevad tähed. Kui aga selline sõltuvus on olemas, peaks kaalumaatriksite asemel

kasutama *peidetud markovi mudeleid (Hidden Markov Models, HMM)*. Käesolevas töös neid ei kasutata ning seetõttu siin nendest pikemalt ei kirjuta. Huvi korral saab peidetud markovi mudelite kasutusest bioinformaatikas lugeda rohkem allikates [Edd98] ja [Kar99].

2.4 Statistilised hüpoteesid

2.4.1 Hüpoteesi mõiste

Üks kõige olulisemaid matemaatilise statistika ülesandeid on kontrollida mitmesuguseid hüpoteese [TM97]. Statistilisi hüpoteese sõnastatakse tavaliselt kas jaotuste või jaotusparameetrite kohta. Käesolevas töös püstitatud hüpotees on jaotuse kohta, seega jaotusparameetrite hüpoteeside tõestust siin ei käsitleta.

Statistilised hüpoteesid sõnastatakse enamasti alternatiivsete hüpoteeside paarina H_0 ja H_1, kusjuures nendest hüpoteesidest saab tõene olla üks ja ainult üks. Hüpoteesi H_0 nimetatakse nullhüpoteesiks. Nullhüpoteesi ei ole võimalik tõestada. Selle vastuvõtmine võib tähendada uuringu jätkumist. Hüpoteesid jaotuse kohta sõnastatakse tavaliselt alljärgnevalt (2.1):

$$H_0 : X \sim F_0(\theta)$$
$$H_1 : X \,!\sim F_0(\theta)$$

(2.1)

Null-hüpotees väidab, et tunnuse kirjeldamiseks sobib mingi konkreetne tõenäosusjaotus $F_0(\theta)$, sisukas hüpotees aga väidab, et see tõenäosusjaotus ei sobi tunnuse kirjeldamiseks. Vastupidist sisu pole hüpoteesidele võimalik anda – tõestada teatud tõenäosusjaotuse sobivust pole matemaatilise statistika vahendite abil võimalik, saab aga tõestada, et teatav tõenäosusjaotus ei sobi tunnuse kirjeldamiseks [Par89]. Praktikas on selliste hüpoteeside tõestamiseks kasutusel mitmed erinevad kriteeriumid, mida ühiselt nimetatakse kooskõlakriteeriumiteks. Käesolevas töös on kasutatud jaotuse ebasobivuse tõestamiseks χ^2 - kriteeriumit, millest on juttu allpool.

2.4.2 Vead hüpoteeside kontrollimisel

Et statistiliste hüpoteeside kontrollimisel tehakse valimi põhjal järeldusi üldkogumi

kohta, tuleb paratamatult arvestada võimalusega, et otsustamisel tekib viga ja tehakse vale järeldus. Statistiliste hüpoteeside kontrollimise teooria põhiliseks iseärasuseks on, et otsustamise juures reguleeritakse vigade esinemise tõenäosusi. Et hüpoteesid H_0 ja H_1 pole samaväärsed, pole seda ka vead. Kokkuvõttes on otsustamise käigus võimalikud järgmised tulemused.

Tegelikkus/Otsustus	***Võetakse vastu H0***	***Võetakse vastu H1***
Kehtib H_0	Õige	1. liiki viga
Kehtib H_1	2. liiki viga	Õige

Statistilised hüpoteesid sõnastatakse reeglina nii, et eriti ebasoovitav on esimest liiki viga (loetakse tõestatuks sisukas hüpotees, kuigi tegelikult on õige nullhüpotees). Seda arvestades konstrueeritakse otsustuse vastuvõtmise eeskiri (nn kriteerium) nii, et esimest liiki vea tõenäosus oleks küllalt väike [TM97].

Esimest liiki vea tegemise suurimat lubatavat tõenäosust nimetatakse olulisuse nivooks ja tema tavaliseks tähiseks on α .

2.4.3 Kvantiil ja täiendkvantiil

Defineerime meeldetuletuseks üle ka kvantiili ja täiendkvantiili mõiste.

Juhusliku suuruse Z jaoks nimetatakse α - kvantiiliks niisugust arvu q_α , mille korral kehtib võrdus

$P(Z < q_\alpha) = \alpha$.

Juhusliku suuruse Z jaoks nimetatakse α - täiendkvantiiliks niisugust arvu $\bar{q}_\alpha$, mille korral kehtib võrdus

$P(Z > \bar{q}_\alpha) = \alpha$ [Par89].

2.4.4 Hii – ruut kriteerium

Põhimõtteline skeem hii-ruut kriteeriumi tuletamiseks on järgmine:

Esimese sammuna jagatakse tunnuse võimalike väärtuste hulk enam-vähem suvaliselt valitud klassideks (olgu need $\Delta_1, \Delta_2, ..., \Delta_k$). Seejärel tehakse iga klassi jaoks kindlaks sinna kuuluvate valimi elementide arv (olgu see $m_1, m_2, ..., m_k$). Siis arvutatakse iga klassi jaoks tõenäosus vaatlustulemuse sellesse klassi sattumiseks tõenäosusjaotuse $F_0(\theta)$ korral (olgu see $p_1, p_2, ..., p_k$). Seda tõenäosust ja valimi mahtu teades saab iga klassi jaoks arvutada "ootuspärase" elementide arvu $n_i = np_i$, kus *n* on elementide koguarv. Arvestades nüüd tõenäosusteooria põhitõde – suure katsete arvu korral ei saa sündmuse tõenäosus ja selle sündmuse suhteline sagedus väga palju erineda – ei tohi mudeli $F_0(\theta)$ sobivuse korral arvude n_i ja m_i erinevus olla väga suur [Par89].

On tõestatud teoreem sobiva statistiku kohta eespool kirjeldatud hüpoteeside kontrollimiseks. See väidab, et sobivaks statistikuks on hii-ruut statistik (2.2):

$$H^2 = \sum_{i=1}^{k} \frac{(m_i - np_i)^2}{np_i} \qquad \textbf{(2.2)}$$

Hii – ruut statistiku kohta on aga teada, et kui kehtib null-hüpotees, siis χ^2 - statistik on χ^2 - jaotusega vabadusastmetega (k – 1). Seega χ^2 - kriteerium näeb välja järgmine:

$H^2 > \bar{h}_\alpha \rightarrow H_1$

$H^2 \leq \bar{h}_\alpha \rightarrow H_0$

Kui $H^2 > \bar{h}_\alpha$, kus $\bar{h}_\alpha$ on χ^2 - jaotuse täiendkvantiil, võtame vastu sisuka hüpoteesi.

Kui $H^2 \leq \bar{h}_\alpha$, jääme null-hüpoteesi juurde.

3. Peatükk

Klasterdamine

Käesolev töö on mõeldud täiendamaks SPEXS'i [Vil02, VKK+03, url:SPEXS] eesmärgiga muuta oluliste mustrite otsimist täpsemaks, vähendada valepositiivseid ja valenegatiivseid. Esimese osa kogu tööst moodustab SPEXS'i väljundi (vt. Tabel 3.1) töötlemine – mustrite klasterdamine.

SPEXS'i tööd võib lühidalt kirjeldada järgmiselt: Tööriistale antakse ette kaks tekstifaili. Ühes on teatud genoomi suvalised alamjärjestused, nn taustafail, ning teises samast genoomist teatud tingimustel valitud alamjärjestused, mida soovitakse lähemalt uurida. SPEXS leiab kõik sellised mustrid, mis esinevad teises failis suhteliselt sagedamini kui taustafailis. Need mustrid on tõenäoliselt seotud funktsiooniga, mille põhjal huvipakkuvad alamjärjestused välja valiti. Näiteks transkriptsioonifaktorite seondumissaite otsides võib esimese failina ette anda terve pärmi genoomi ning teises failis selle kõik promootorpiirkonnad. Väljastatavate mustrite seas on sellisel juhul tõenäoliselt ka seondumissaidid.

Näide SPEXS'i väljundist on näha tabelis 3.1.

1. G.GATGAG.T	62/75	1:39/49	2:23/26	R:17.3026	BP:1.12008e-37
2. G.GATGAG	89/110	1:45/60	2:44/50	R:10.436	BP:1.61764e-34
3. GATGAG.T	124/148	1:52/70	2:72/78	R:7.36961	BP:2.79148e-33
4. TG.AAA.TTT	132/145	1:53/61	2:79/84	R:6.84578	BP:1.83509e-32
5. AAAATTTT	200/231	1:63/77	2:137/154	R:4.69239	BP:1.19109e-30
6. TGAAAA.TTT	104/114	1:45/53	2:59/61	R:7.78277	BP:3.86086e-29
7. AAA.TTTT	343/537	1:79/145	2:264/392	R:3.05349	BP:5.66833e-29
8. G.AAA.TTTT	135/156	1:51/62	2:84/94	R:6.19534	BP:5.69933e-29
9. TG.GATGAG	49/57	1:30/35	2:19/22	R:16.1117	BP:9.35765e-28
10. TG.AAA.TTTT	86/91	1:40/43	2:46/48	R:8.87311	BP:1.1124e-27

Tabel 3.1

Failis, millest otsisime mustreid, oli 98 sekventsi, taustafailis 6423. Esimeses veerus tuuakse ära vastava sekventsi esinemiste arv mõlemas failis kokku (esimese mustri puhul 62-s erinevas sekventsis 75 esinemist). Teises veerus on toodud esinemissagedus huvialuses failis (esines 39-s sekventsis 49 korda) ning kolmandas esinemissagedus taustafailis. Neljas veerg näitab, kui mitu korda esines antud sekvents suhteliselt huvi pakkuvas failis rohkem kui taustafailis ning viimane veerg näitab suhet binoomtõenäosuse järgi (tõenäosus, et saadud tulemus võis tekkida juhuslikult).

SPEXS’i väljundis on aga palju liigset informatsiooni. Lisaks mustrile G.GATGAG.T tuuakse eraldi välja ka selle alammustrid G.GATGAG ja GATGAG.T. Kokkuvõttes on raske aru saada, millised nendest mustritest tegelikult olulised on, seega tuleb väljundmustreid klasterdada.

Klasterdamise jaoks on erinevaid meetodeid palju. Üks võimalus on arvutada dünaamilise programmeerimise meetodil mustrite vahelised kaugused ning kui kaugus ületab etteantud lävendi, lugeda mustrid erinevaks. Käesolevas töös läheneme aga asjale teise nurga alt - kasutame mustrite klasterdamiseks kaalumaatrikseid. Sama funktsiooniga sekventsid erinevad sageli teineteisest tänu mutatsioonidele ning bioloogiliste mustrite klasterdamisel oleks õigem arvestada mitte ainult järjestust “GTAACGTTT”, vaid ka sellele lähedasi järjestusi, mis tõenäoliselt sama funktsiooni omavad.

Meie algoritmi üldine kuju on järgmine:

1. Sorteerime sisendmustrid
2. Iga mustri puhul leiame kõik sellega sarnased järjestused
3. Leitud järjestuste põhjal genereerime iga mustri jaoks kaalumaatriksi
4. Võrdleme kaalumaatrikseid omavahel

Mustrite sorteerimine on oluline, kuna see mõjutab otseselt klasterduse tulemust. Eespool toodud väljundi näites on mustrid sorteeritud binoomtõenäosuse järgi.

Igale mustrile sarnased järjestused leitakse programmi *fagrep* abil. Fagrep on Shift-or algoritmi kasutav programmi *agrep* (approximated grep [SFW83]) implementatsioon, mis võimaldab samuti ligikaudselt otsida, andes ette maksimaalse erinevuste arvu. Sellega leitakse kõik hetkel kasutatava mustriga sarnased järjestused taustafailist ning moodustatakse nende põhjal kaalumaatriksid.

Järgmine probleem seisneb selles, kuidas kaalumaatrikseid omavahel võrrelda. Peab arvestama samade probleemidega nagu sõnede vahelise võrdluse puhulgi, st. peab arvestama võimalusega, et maatriksid on omavahel nihkes. Kuna põhimõttelt on probleem väga sarnane kahe sõne vahelise kauguse arvutamisega, kasutame maatriksite erinevuse arvutamiseks dünaamilise programmeerimise meetodit, st. leiame teisenduskauguse. Tähtede võrdlemise asemel võrdleme veergusid (horisontaalne

maatriks – tähed on esimeses veerus) või ridu (vertikaalne maatriks – tähed on esimeses reas) omavahel.

Veergude vahelist erinevust arvutame Eukleidese kauguse abil. Olgu võrreldavad maatriksid *A* ja *B*, siis horisontaalmaatriksi *A* veeru *i* ja maatriksi *B* veeru *j* erinevuse leiame järgmise valemi abil:

$$d(A_i, B_j) = \sqrt{\sum_{k=1}^{m} (A_{k,i} - B_{k,j})^2} \ ,$$

kus *m* – ridade e. tähtede arv (DNA puhul 4).

Maatriksite vahelist kaugust arvutame dünaamilise programmeerimise meetodil (3.1). Samasugune algoritm on peatükis 4 toodud *globaalse joonduse* probleemi all.

$$D(i, j) = \min \begin{bmatrix} D(i-1, j) + kustutamise\_tasu, \\ D(i, j-1) + lisamise\_tasu, \\ D(i-1, j-1) + d(i, j) \end{bmatrix}$$

(3.1)

Kui teisenduskaugus ületab etteantud lävendi, loetakse maatriksid, ning seega esialgsed mustrid, erinevaks. Lävend on arvuline suurus, mille antud ülesandes leidsime katse-eksituse meetodil, st vaatasime, milline lävend silma järgi kõige paremini sarnased mustrid klasterdab. Kui mustrid loetakse etteantud parameetrite põhjal sarnasteks, lisatakse teise maatriksi moodustanud muster esimese maatriksi moodustanud mustri klastrisse ning tema maatriks kustutatakse. Tulemuseks on mustrite klastrid massiivide kujul. Vt. algoritm 3.1.

Algoritm 3.1 *Mustrite klasterdamine kaalumaatriksite baasil*

Sisend: *SPEXS väljund eespool toodud kujul, mustrite arv n, mustrite tähised P, etteantud lävend t maatriksite võrdluseks, etteantud lävend l järjestuste s võrdluseks*

Väljund: *Massiivid* $C_{1..k}$ *erinevate mustrite klastrite salvestamiseks*

Meetod:

1. k = 1;
2. **for** (i = 1; i ≤ n; i++)
3. *Leiame kõik mustriga* P_i *sarnased järjestused* $S = \{s \mid d(s, P_i) \le l\}$
4. *Moodustame nende põhjal kaalumaatriksi* W_i
5. **for** (j = 1; j ≤ k; j++)
6. **if** (d(W_i, M_j) < t) //*Maatriksid loetakse sarnasteks*
7. **push** (P_i, C_j) //*Paigutatakse sarnased mustrid samasse klastrisse*
8. **else** //*Maatriksid on erinevad*
9. **push** (P_i, C_k)
10. $M_k = W_i$ //*Salvestatakse klastrid moodustanud mustrite maatriksid*
11. k++;
12. **return** *C* //*Tagastatakse klastrite massiiv*

Näide 3.1: SPEXS'i väljundmustrid tabelis 1.1 grupeeritakse kaheks klastriks:

```
G.GATGAG.T          TG.AAA.TTT
G.GATGAG            AAAATTTT
GATGAG.T            TGAAAA.TTT
TG.GATGAG           AAA.TTTT
                    G.AAA.TTTT
                    TG.AAA.TTTT
```

Näide 3.2: Klasterdades 718 mustrit lävendiga, mis kõige paremini sarnased mustrid grupeerib, saime 16 mustrite klastrit. Lävendit suurendades väheneb klastrite arv, kuid samuti väheneb positsiooniti tähtede skooride vahe.

Näide 3.3: Klasterdades 1498 mustrit, saime 25 klastrit.

Tulles tagasi töö eesmärgi juurde, milleks oli SPEXS väljundi modifitseerimine nii, et alles jääks vaid paar mustrite gruppi, on peale mustrite klastrite moodustamist vaja neid kuidagi üheselt esitada. Üks variant on väljundisse jätta vaid mustrid, mille põhjal

klastrid tekitati, ehk siis meie poolt valitud kriteeriumi järgi kõige olulisemad mustrid. Siiski oleks parem kujutada väljundis tervet klastrit täpsemalt iseloomustavat mudelit. Selleks otsustasime igas klastris mustrid joondada (kuna mustrid on erineva pikkusega) ning saadud joonduse põhjal koostada logo. Järgmine peatükk selgitabki, kuidas me lahendasime mustrite joondamise probleemi.

4. Peatükk

Mitmene joondamine

Käesolev peatükk käsitleb mitmest joondamist. Kõigepealt selgitame joondamise mõistet ning räägime erinevatest joondamise meetoditest. Seejärel anname seletuse mitmese joondamise mõistele ning kirjeldame detailselt mitmese joondamise probleemi lahendamiseks kasutatud algoritmi.

4.1 Joondamine

Üks olulisemaid meetodeid geeni (või tema poolt kodeeritud valgu) bioloogilise funktsiooni määramiseks on DNA ja valgu andmebaasides sarnaste sekventside otsimine, kuna sarnase ehitusega sekventsid on sageli ka sarnase funktsiooniga. Kõige paremini on sekventside sarnasused näha nende *joondusest* (ingl. k. alignment) – kahe võrreldava järjestuse esitus, mis toob välja nende statistiliselt kõige sarnasemad piirkonnad. See muudab joondamise bioinformaatika üheks tähtsamaks probleemiks.

Joondamismeetodid võib jagada kahte põhigruppi: täpsed algoritmid, mis garanteeritult arvutavad optimaalse teisenduskauguse (Needleman-Wunsch algoritm, Smith-Waterman algoritm), ning väga kiired heuristilised meetodid, mis optimaalsust ei garanteeri (FASTA, BLAST) [Pea01]. Käesolevas töös on kasutatud ainult esimesi algoritme ning teistest pikemalt ei räägita. FASTA ja BLASTi statistilise tausta kohta saab pikemalt lugeda allikas [EG01].

4.1.1Globaalne joondamine

Nii globaalne kui ka allpool vaadeldav lokaalne joondamine baseeruvad suurel määral *ligikaudse otsimise probleemil*. Seepärast selgitan kõigepealt ligikaudse sobitamisega seotud mõisteid.

> Definitsioon. Ligikaudse otsimise probleemiks nimetatakse ülesannet, mis peab leidma erinevuste arvu (*teisenduskauguse*) mustri P (pikkusega *m*) ja teksti T iga *m*-pikkusega alamsõne vahel [ALP00].

> Definitsioon. Kahe sõne vaheliseks teisenduskauguseks nimetatakse minimaalset arvu teisendusoperatsioone – lisamisi, kustutamisi ja asendusi, mida on vaja

esimese sõne muutmiseks teiseks sõneks [Gus99].

Teisenduskauguse probleem on kahe sõne vahelise optimaalse teisenduskauguse arvutamine. Kui teisenduskauguse probleem on lahendatud, saab tulemuse põhjal moodustada kahe sõne globaalse *joonduse*:

Definitsioon. Sõnede S_1 ja S_2 globaalne joondus saadakse vajalike lünkade lisamisel S_1 ja S_2 sisse ning äärtesse ja seejärel sõnede paigutamisel teineteise kohale, nii et igale tähemärgile ja lüngale ühes sõnes vastab tähemärk või lünk teises sõnes [Gus99].

Näiteks toon sõnede "*asdflkf*" ja "*agsdfkj*" globaalse joonduse:

a	–	*s*	*d*	*f*	*l*	*k*	*f*
a	*g*	*s*	*d*	*f*	–	*k*	*j*

Dünaamilise programmeerimise meetodit kasutades saab teisenduskauguse probleemi üsna lihtsalt lahendada ajaga $O(mn)$. Sõnede S_1 ja S_2 puhul tähistame teisenduskauguse $S_1[1..i]$ ja $S_2[1..j]$ vahel $D(i, j)$. Algoritm $D(i, j)$ arvutamiseks on järgmine:

$$D(i,0) = i$$
$$D(0, j) = j$$
$$D(i, j) = \min \begin{bmatrix} D(i-1, j)+1, \\ D(i, j-1)+1, \\ D(i-1, j-1)+t(i, j) \end{bmatrix}$$

kus $t(i, j) = 1$, kui $S_1(i) \neq S_2(j)$

$t(i, j) = 0$, kui $S_1(i) = S_2(j)$ [5].

Element saadud maatriksis kohal *(m, n)* ongi sõnede S_m ja S_n teisenduskaugus.

Kui teisenduskaugus on käes, tuleb globaalse joonduse saamiseks teha nn. *traceback*: läbitakse sama tee teistpidi, st. alustatakse elemendist $D(m,n)$ ja leitakse, mis teed pidi selleni jõuti:

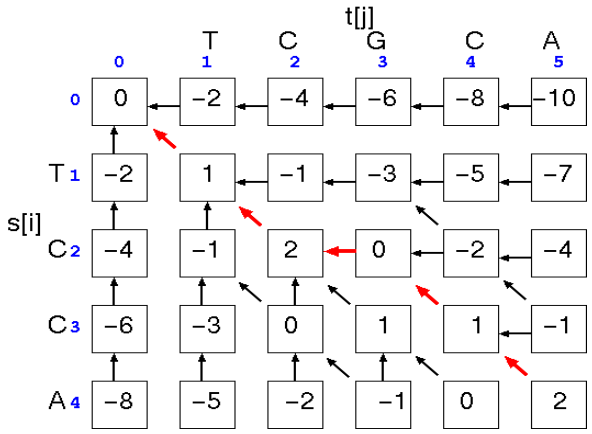
Kui $D(i, j) = D(i, j-1)+1$, lisatakse tühik vertikaalsesse sõnesse kohta *i* (lisamine).

Kui $D(i, j) = D(i-1, j)+1$, lisatakse tühik horisontaalsesse sõnesse kohta *j*

(kustutamine).

Kui $D(i, j) = D(i-1, j-1) + t(i, j)$, asetatakse vastavad tähed lihtsalt teineteise kohale (asendamine).

Näide [url:NTU]:



Globaalne joondus antud näite puhul on:

T C G C A
T C – C A

4.1.2 Lokaalne joondamine ja Smith – Waterman algoritm

Paljudel juhtudel pole sõned terves pikkuses omavahel väga sarnased, kuid võivad sisaldada alamsõnesid, mis seda on. Ülesandeks on leida suure sarnasusega piirkonnad mõlemas järjestuses. Seda kutsutakse *lokaalse joondamise* e. *lokaalse sarnasuse probleemiks* ja on formaalselt defineeritud järgmiselt:

> Definitsioon. Olgu meil antud kaks järjestust S_1 ja S_2. Leia alamstringid α ja β vastavalt järjestustest S_1 ja S_2, mille sarnasus $V(\alpha, \beta)$ (optimaalne globaalse joondamise väärtus) on maksimaalne üle kõigi alamstringipaaride järjestustest S_1 ja S_2. Tähistagu v^* lokaalse joondamise probleemi optimaalse lahenduse väärtust [Gus99].

Võrreldes lokaalset ja globaalset joondamist, võib öelda, et globaalset joondamist on mõttekas kasutada juhul, kui võrreldavad järjestused on kogu ulatuses sarnased. Lokaalset joondamist on aga soovitav kasutada, kui järjestused kogu ulatuses väga sarnased pole, küll aga leiduvad neis alamsõned, mis on struktuurilt teineteisega väga

lähedased.

Olgu meil järjestused S_1 ja S_2 pikkustega vastavalt *n* ja *m*. Sellisel juhul saab lokaalse joondamise probleemi lahendada *O(nm)* ajaga, kui kasutada Temple Smith ja Michael Waterman'i poolt kasutusele võetud algoritmi, mida järgnevalt kirjeldan.

Enne lokaalse joonduse arvutamise juurde asumist, anname *lokaalse sufiksi joonduse probleemi* definitsiooni:

> Definitsioon. Olgu meil indeksid $i \le n$ ja $j \le m$. Lokaalne sufiksi joondamise probleem on leida sufiksid α alamsõnest $S_1[1..i]$ ja β alamsõnest $S_2[1..j]$ nii, et $V(\alpha, \beta)$ on maksimaalne üle kõikide sufiksite paaride. Võtame optimaalse lokaalse sufiksi joonduse väärtuse tähiseks indeksite *i* ja *j* puhul *v(i, j)* [Gus99].

Järgmised teoreemid näitavad, kuidas on omavahel seotud lokaalse joonduse ning lokaalse sufiksi joonduse probleemid:

> Teoreem 2.4.1. $v^* = \max[v(i, j) : i \le n, j \le m]$

> Toreem 2.4.2. Kui *i'* ja *j'* on indeksid, mille puhul *v(i, j)* on maksimaalne üle kõigi *i, j* paaride, siis alamsõnede paar, mis lahendab lokaalse sufiksi joonduse probleemi, lahendab ka lokaalse joonduse probleemi.

Lokaalse sufiksi joonduse probleemi lahendus on peidus järgmises teoreemis:

> Teoreem 2.4.3. $\forall i, j : v(i, 0) = 0, v(0, j) = 0$

$\forall i > 0, j > 0:$

$$v(i, j) = \max \begin{bmatrix} 0, \\ v(i-1, j-1) + s(S_1(i), S_2(j)), \\ v(i-1, j) + s(S_1(i), \_), \\ v(i, j-1) + s(\_, S_2(j)) \end{bmatrix}$$

kus $s(S_1(i), S_2(j)) = samasus\_tasu$, kui $S_1(i) = S_2(j)$,

$s(S_1(i), S_2(j)) = erinevus\_tasu$, kui $S_1(i) \neq S_2(j)$,

$s(S_1(i), \_) = kustutamise\_tasu$,

$s(\_, S_2(i)) = lisamise\_tasu$.

See teoreem annab eeskirja lokaalse sufiksi joonduse maatriksi arvutamiseks. Lokaalse joonduse saab kätte, kui teha *traceback* alates maatriksi maksimaalsest elemendist.

Alltoodud näites on joondatud lokaalselt HEAGAWGHEE ja PAWHEAE ning nende optimaane lokaalne joondus on välja toodud tabeli all [DEKM99]:

		H	E	A	G	A	W	G	H	E	E
	0	0	0	0	0	0	0	0	0	0	0
P	0	0	0	0	0	0	0	0	0	0	0
A	0	0	0	5	0	**5**	0	0	0	0	0
W	0	0	0	0	2	0	**20** ←	**12** ←	4	0	0
H	0	10 ←	2	0	0	0	12	18	**22** ←	14 ←	6
E	0	2	16 ←	8	0	0	4	10	18	**28**	20
A	0	0	8	21 ←	13	5	0	4	10	20	27
E	0	0	6	13	18	12 ←	4	0	4	16	26

AWGHE
AW-HE

From Durbin et al. 1998

Molekulaarbioloogia-alases kirjanduses nimetatakse lokaalset joondust sageli Smith-Waterman joonduseks ning globaalset joondust Needleman-Wunsch joonduseks. Nende tähendus on aga tegelikult erinev. Needleman'i ja Wunsch'i poolt tutvustatud lahendus on kuup-keerukusega ning seda kasutatakse harva. Seega "Needleman-Wunsch" viitab globaalse joonduse probleemile. "Smith-Waterman" aga viitab enamasti eespool toodud algoritmile, mis tegeleb lokaalse joonduse probleemi lahendamisega [Gus99].

4.2. Mitmene joondamine

Selleks, et üheselt määratleda mustrite klastrit, võib nende põhjal moodustada regulaaravaldise, täpsemalt kirjeldab klastrit aga kaalumaatriks. Nagu eespool mainitud, on mustrid erineva pikkusega ja seega tekib järgmine probleem:

Olgu meil järgmised mustrid:

AAAAATTT
AAATTT
AAAATTT

Nende põhjal moodustatud loendusmaatriks näeb välja järgmine:

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

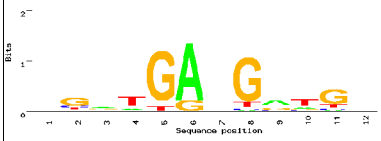
Kuid see maatriks ei kajasta mustreid nii nagu meil vaja. Selle profiili kohaselt võib nende mustrite hulka sobitada ka mustri AAATATTT, kuid eelpool toodud mustrid on kujul A{3, 5}TTT. Seepärast peab mustrid eelnevalt joondama.

Käesolevas töös kasutatakse joondamisel dünaamilise programmeerimise põhimõtet. Joondamise meetoditest on globaalsele joondamisele eelistatud lokaalset, kuna siin on oluline, et sarnased piirkonnad võimalikult suures ulatuses kohakuti jääks. Mõlema meetodi kohta on põhjalikumalt kirjutatud peatükis 2. Käesolevas töös kitsendatakse lokaalse joondamise meetodit sellega, et joondatavatesse sõnedesse lünki ei tekita.

Globaalse ning lokaalse joondamise põhierinevust mitmesel joondamisel selgitab järgmine näide.

Olgu meil vaja joondada järgmised sõned (vt. tablel 4.1):

Tabel 4.1.

Esialgsed sõned	***Globaalselt joondatud***	***Lokaalselt joondatud***
G.GATGAG GATGAG.TG CGATGAG.T TG.GATGAG GATGAG.TGA ATGAG.T GATGAGCT GATGAGC ATGAG.TG G.GATGAGC GATGAGATG G.TGAGATG	G-.GATGA-G- GATGA-G.TG- CGATGA-G.T- TG-.GATGA-G GATGA-G.TGA -ATGA-G.T-- GATGA-GCT-- GATGA-GC--- -ATGA-G.TG- G-.GATGA-GC GATGA-GATG- G.TGA-GATG-	-G.GATGAG---- ---GATGAG.TG- --CGATGAG.T-- TG.GATGAG---- ---GATGAG.TGA ----ATGAG.T-- ---GATGAGCT-- ---GATGAGC--- ----ATGAG.TG- -G.GATGAGC--- ---GATGAGATG- ---G.TGAGATG-
Joondatud mustrite graafiline esitus:		

Nagu näha tabelist 4.1, on lokaalse joonduse puhul selgelt näha kõigi sõnede sarnane tüvi. Logod on genereeritud programmi SEQLOGO abil [VKK+03, url:SEQLOGO].

Kuna antud probleemi puhul on omavahel vaja joondada enamasti rohkem kui kaks mustrit, on tegemist mitmese joondamisega. Selleks on välja pakutud palju erinevaid võimalusi, alates dünaamilise programmeerimise meetodi laiendamisest (iga mustriga lisandub üks dimension ning seetõttu on keerukus $O(n^k)$, kus n – mustri pikkus) sufiksipuude abil joondamiseni [Gus99]. Lühikesi järjestusi on joondatud ka pikima ühise alamsõne abil [Vil02]. See meetod töötab siis, kui joondatavates järjestustes leidub suhteliselt pikk sarnane alamsõne. Vastasel juhul valitakse ühine alamsõne väga lühike (näiteks 2 tähte) ning joondus kaotab mõtte.

Käesolev töö on mõeldud lühikeste järjestustega opereerimiseks (5 – 20 nukleotiidi). Soovisime kasutada lokaalset joondamist nii, et sekventsi sisse ühtegi lünka ei lisata, seega struktuuri ei muudeta. Kõigepealt otsustasime kasutada joondamisel abistringi, mis kajastaks kõigi eelnevalt joondatud mustrite konsensust. Sellel on aga teatud negatiivseid külgi ning lõpuks otsustasime hoopis abimaatriksi kasuks.

Õige maatriksi tüübi leidmiseks uurisime, mis tingimused peavad olema täidetud järgmise näite abil:

Olgu meil sekventsid

G A T G A G . T
A G A T G A T
G A G A T G A G

Ja olgu meil joondatud kaks esimest sekventsi:

– G A T G A G . T
A G A T G A T – –

Nende põhjal koostatud loendusmaatriks näeb välja järgmine ('.' annab igale tähele 0.25, '_' ei mõjuta maatriksit):

	1	2	3	4	5	6	7	8	9
A	1	0	2	0	0	2	0	0.25	0
C	0	0	0	0	0	0	0	0.25	0
G	0	2	0	0	2	0	1	0.25	0
T	0	0	0	2	0	0	1	0.25	1

On vaja, et abimaatriksis kehtiksid järgmised tingimused:

1. Tähe 'A' kaal on suurem veerus 3 kui veerus 1, kuna seda on mõjutanud rohkem sekventse.
1. Tähe 'T' kaal on suurem veerus 4 kui veerus 9 ning suurem veerus 9 kui veerus 7.

Ei loendusmaatriks ega tõenäosusmaatriks rahulda neid tingimusi, samuti mitte kaalumaatriks. Sellepärast konstrueerisime uue maatriksi tüübi, mida edaspidi nimetame *olulisusmaatriksiks*. Selleks korrutatakse loendusmaatriksi ning selle põhjal konstrueeritud tõenäosusmaatriksi vastavad elemendid omavahel. Saadud maatriks rahuldab eespool toodud tingimusi.

Saadud olulisusmaatriksi abil joondamine käib järgmiselt: leitakse dünaamilise programmeerimise abil, mis positsioonile sekvents maatriksis kõige paremini sobib. Meie algoritm näeb üldjoontes välja järgmine:

1. Olulisusmaatriksisse lisatakse esimene muster

2. Teine muster joondatakse maatriksiga ning lisatakse seejärel samuti maatriksisse,

vajadusel lisades verge

3. Korratakse protseduuri kõigi ülejäänud mustrite jaoks

Dünaamilise programmeerimise maatriks koostatakse järgmiselt:

$$\forall i,\, j : v(i,0) = 0, v(0,\, j) = 0$$

$$\forall i > 0,\, j > 0 :$$
$$v(i,\, j) = \max \begin{bmatrix} 0, I_{S_i,\, j} = 0, \\ v(i-1,\, j-1) + I_{S_i,\, j}, I_{S_i,\, j} > 0 \end{bmatrix}$$

Kus I - olulisusmaatriks,

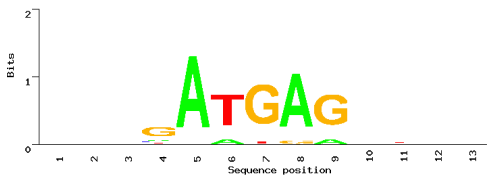
S – joondatav sekvents.

Kui lokaalse joondamise maatriks on valmis, joondatakse muster maatriksiga nii, nagu teise sõnegagi, st. leitakse maksimaalne element ning selle järgi joondatakse omavahel maatriks ja uus muster. Kui järjestus ei mahu maatriksisse, lisatakse loendusmaatriksisse vastavasse äärde vajalik arv uusi verge, mis esialgu koosnevad 0-dest. Seejärel lisatakse uus muster loendusmaatriksisse, st vastavates lahtrites liidetakse olemasolevale arvule 1. Joondusmaatriksi põhjal genereeritakse uus tõenäosus- ning seejärel uus olulisusmaatriks ning korratakse sama protseduuri uue maatriksi puhul. Seda kirjeldab täpsemalt algoritm 4.3.

Näide 4.1: Näites 3.1. toodud mustrite klastrite joondus näeb meie algoritmi järgi välja selline:

```
-G.GATGAG.T          TG.AAA.TTT-
-G.GATGAG--          ---AAAATTTT
---GATGAG.T          TGAAAA.TTT-
TG.GATGAG--          ---AAA.TTTT
                     -G.AAA.TTTT
                     TG.AAA.TTTT
```

Näide 4.2: Selles näites on joondatud erinevamaid mustreid. Tabeli esimeses veerus on toodud joondamata mustrid, teises veerus nende joondus ning kolmandas joonduse põhjal genereeritud logo:

A.GAGATG	----A.GAGATG	
A.ATGAG.T	---A.ATGAG.T-	
AT.AGATG	----AT.AGATG	
CG..GAGC	--CG..GAGC--	
GAT.A.ATG	---GAT.A.ATG	
ATGA.ATG	----ATGA.ATG	
GC.AT.AG	-GC.AT.AG---	
GA.G.GAT	---GA.G.GAT-	
GAGA.GAG	-GAGA.GAG---	
ATGA..TG	----ATGA..TG	
AT.AG.TG	----AT.AG.TG	
GATGA.A.G	---GATGA.A.G	
CG.TG.G	--CG.TG.G---	
GAA.AA.TT	---GAA.AA.TT	
C.ATGAGC	--C.ATGAGC--	
CGA.G.G	--CGA.G.G---	
G.TGA.AT	---G.TGA.AT-	
GATGA.C	---GATGA.C--	
GAT..GCT	---GAT..GCT-	
GC.A.GAG	-GC.A.GAG---	
A..GATGAG	A..GATGAG---	

Algoritm 4.3 *Mustrite joondamine abimaatriksi abil*

Sisend: *Joondamata mustrid* $P_{1..n}$

Väljund: *Muudetud mustrite massiiv* $M_{1..n}$, *kus vajalikesse kohtadesse on lisatud lünki*

Meetod:

```
1.  function Main
2.      global C, I;
3.      C_{m*n} = countmatrix( P_1 )
4.      I_{m*n} = importance_matrix(C)
5.      for (i = 2; i ≤ n; i++)
6.          M_i = Align ( P_i , I)
7.      return M
8.
9.  function Align ( P_i , abistring)
10.     Moodusta lokaalse joonduse maatriks
11.     Leia lokaalse joonduse maatriksist suurima elemendi koordinaadid (x_pos, y_pos)
12.     if (x_pos > y_pos)
13.         Lisa x_pos-y_pos lünka mustri M_i algusesse
14.     else if (x_pos < y_pos)
15.         Lisa y_pos-x_pos veergu maatriksi algusesse
16.     if (length( P_i ) + x_pos – y_pos > n) //uus sõne pikem kui praegune maatriks
17.         Lisa veerge maatriksi lõppu
18.     else if (length( P_i )+ x_pos – y_pos < n) //uus muster lühem kui maatriks
19.         Lisa tühikud mustri M_i lõppu
20.     renew (C, P_i ) //Lisa loendusmaatriksisse viimane joondatud muster
21.     renew (I) //Muuda olulisusmaatriksit
22.     return M_i
```

Lõpptulemuse (joondatud sekventside) põhjal moodustatakse kaalumaatriks, ning kaalumaatriksist saab olemasolevate programmide abil moodustada logo – maatriksi graafilise esituse.

5. Peatükk

Maatriksite sobitamine tekstile

Praeguseks on olemas vajaliku tööriista esimene pool: mustrid on klasterdatud ja nende põhjal on moodustatud kaalumaatriksid, st. on leitud paar olulist motiivi. Järgmisena on vaja rakendust, mis leiaks tekstist üles leitud motiividele kõige täpsemini vastavad alamsõned. Kuna motiivid on esitatud kaalumaatriksitena, on meil tegemist maatriksite sobitamisega tekstile. Peatüki esimeses osas on toodud ära erinevaid sobitamise meetodeid. Teises osas tuletatakse meelde prioriteedijada põhimõtteid. Kolmas osa keskendub praktilise tööga seotud probleemidele.

5.1 Sobitamise meetodid

Maatriksi sobitamisel tekstile on eesmärgiks leida alamstringe, mida maatriks kõige paremini kirjeldab ehk nn. signaale. Selleks libistatakse maatriks üle teksti ning igale positsioonile antakse skoor. Kui skoor on piisavalt hea, loetakse vastavat alamsekventsi signaaliks. Skoor leitakse järgmise valemi abil (5.1):

$$score_i = \sum_{j=i}^{i+j-1} W_{S_j, j} \,, \tag{5.1}$$

kus W – maatriks,

S – tekst.

Sobitamist on võimalik kiiremaks muuta, kui on ette antud konkreetne lävend. Järgnevalt räägin lühidalt kolmest levinud sobitamise meetodist [WNB00].

5.1.1 Jõumeetod

Jõumeetodi puhul arvutatakse igal positsioonil alamsekventsi skoor ning võrreldakse seda etteantud lävendiga. Kui skoor ületab lävendi, on tegemist positiivse signaaliga.

5.1.2 Ettevaatav skoorimine

Sobitamist on võimalik märgatavalt kiirendada. Selleks võtan kasutusele abimassiivi Z(m), kus m on maatriksi veergude arv. Z elemendid arvutan valemi (5.2) järgi:

$$Z^{(j)} = \sum_{k=j+1}^{J} \max S_k(a)$$

(5.2)

St. massiiv Z salvestab iga veeru kohale skoori, mida maksimaalselt veel võimalik saada on. Nüüd saab sobitamist kiirendada nii, et iga skoori arvutamisel kontrollitakse igal sammul, kas etteantud lävendit on üldse võimalik enam saavutada. Kui ei, liigutakse tekstis edasi.

5.1.3 Permuteeritud ettevaatav skoorimine

Permuteeritud ettevaatavat skoorimist on veelgi võimalik kiirendada järjestades maatriksi veerud ümber. Iga veeru jaoks saame arvutada maksimaalse elemendi (5.3):

$$M_j = \max_a S_j(a)$$

(5.3)

ning keskväärtuse (5.4):

$$E_j = \sum_a S_j(a) q_a$$

(5.4)

Mida suurem on nende kahe väärtuse vahe, seda rohkem mõjutab vastav veerg lõppskoori. Seega järjestatakse veerud ümber kahanevalt $|E_i - M_i|$ järgi. Sobitamisel peab seda muutunud järjekorda muidugi arvestama ning see võtab teatud ajakulu, kuid mahukates ülesannetes on see meetod ajaliselt kasulik.

5.2 Kahendkuhi e. prioriteedijada

Kui on vaja leida n parimat esinemist etteantud järjestustest, on otstarbekas kasutada selleks prioriteedijada. Järgnevalt kirjeldan meeldetuletuseks prioriteedijada e. kahendkuhja põhimõtet.

Kahendkuhi ehk lihtsalt kuhi on kompaktne kahendpuu, milles ühegi tipu võti pole suurem kui tema ülemuse võti [Kih03]. Seega on juureks suurima võtmega element. Käesolevas töös kasutatakse kahendkuhja pööratud varianti e. pöördkuhja, kus juureks on

vähima võtmega element.

Kuhja sobivaimaks kujutusviisiks on järjestikpaigutus kahendpuu tasemete kaupa: n-tipulise kuhja tipud moodustavad massiivi $a_1, a_2, ..., a_n$, kus a_1 on juur, a_2 ja a_3 on juure alluvad (st. esimese taseme tipud), seejärel tulevad kõik teise taseme tipud jne.; element a_n vastab viimase taseme kõige parempoolsemale tipule.

Vaadeldava tipu a_k korral on alluvate ja ülemuse indeksiteks

vasak(k) = 2k;

parem(k) = 2k + 1;

ülemus(k) = $\left\lfloor \frac{k}{2} \right\rfloor$ [Kih03];

Kuhja lisamine on lihtne ning ajaline keerukus on halvimal juhul O(log *n*).

Käesolevas töös kasutatakse prioriteedijada *n* suurima elemendi leidmiseks ning salvestamiseks. Selleks võrrledakse uut elementi kõigepealt kuhja juurega. Kui uus element on juurest suurem, lisatakse see jada lõppu ning viiakse rekursiivselt õigesse kohta, nii et kehtiks pöördkuhja tingimus – ükski alluv pole ülemusest väiksem. Selleks rakendatakse järgmist funktsiooni [AU00]:

```
void bubbleUp (int A[], int i)
{
        if (i > 1 && A[i] < A[i/2]){
                swap (A[i], A[i/2]);
                bubbleUp(A, i/2);
        }
}
```

Selgitama peab ka seda, et jada esimene element asub kohal 1, mitte kohal 0 nagu mitmetes programmeerimiskeeltes tavaks on. Vastasel juhul ei kehtiks tingimus, et ülemuse alluvad on indeksitega 2k ning 2k+1.

5.3. Kaalumaatriksite sobitamine tekstile

Kaalumaatriksite sobitamine tekstile on põhimõttelt väga sarnane ligikaudse otsimisega. Nimelt soovitakse leida etteantud tekstist alamsõned, mida etteantud maatriks kõige paremini kirjeldab.

Tööriistale esitatavad nõudmised on järgmised:

Andes sisendiks maatriksi ja tekstifaili,

1. leida kõik alamstringid, mille sobivusskoor ületab etteantud lävendi
2. leida n sobivaimat alamstringi
3. leida kõik head esinemised

Kõik kolm nõudmist on üldjoontes sarnased. Esimest saab lahendada punktis 5.1 toodud sobitamisvõtetega (Algoritm 5.1). *n* parima ning kõikide heade esinemiste leidmisele on pühendatud alampeatükid 5.3.1 ja 5.3.2.

Algoritm 5.1 *Kõigi lävendist parema skooriga alamstringide leidmine*

Sisend: *Maatriks M[rows][columns], tekst T, lävend l*

Väljund: *Struktuuride massiiv best_scores, kus salvestatakse skoor ning positsioon*

Meetod:

```
1.  function Main
2.      double max_array = maxarray(M);
3.      match_1(M, T, l);
4.  function maxarray (maatriks M)
5.      double max_elements[columns];
6.      for (j = 0; j < columns; j++)
7.           max_elements[j] = 0;
8.           for (i = 0; i < rows; i++)
9.                if (M[i][j] > max_elements[j]) max_elements[j] = M[i][j];
10.    double max_array[columns];
11.    for (i = 1; i < columns; i++)
12.         max_array[i-1] = 0;
13.         for (j = i; j < columns; j++)
14.              max_array[i-1] += max_elements[j];
15.    return max_array;
16. function match_1 (maatriks M, tekst T, lävend l)
17.    double sum = 0.0;
18.    int a = 0;
19.    struct best best_scores;
20.    for (i = 0; i < length(l) – columns + 1; i++)
21.         for (j = 0; j < columns; j++)
22.              sum += M[T[i+j]][j]
23.              if (!(sum + max_array[j+1] > l))
24.                   break
25.         best_scores[a].score = sum
26.         a++
27.   return best_scores
```

5.3.1 *n* parima esinemise leidmine

Esimene mõte, kuidas *n* parimat skoori asukohtadega salvestada, oli luua skooride järgi järjestatud struktuuride massiiv. Esialgu täidetakse massiiv n esimese skooriga, need järjestatakse, ning iga järgmise skoori puhul võrreldakse seda massiivi madalaima skooriga. Kui skoor ületab massiivi madalaimat, leitakse uuele skoorile kahendotsimise teel koht ning ülejäänusid nihutatakse ühe võrra paremale. Praktikas töötab see suhteliselt kiirelt kui n on väike, 10 000 ja 100 000 massiivi puhul aga väga aeglaselt.

Teine võimalus on kasutada viitade liste (ingl. k. *linked lists*). On aga ilmne, et see töötab veel aeglasemalt, kuna õige koha leidmiseks peab alati algusest peale kõik läbi vaatama.

Kolmas ja kõige efektiivsem meetod antud probleemi puhul on pööratud prioriteedijada, kus esimesel kohal on massiivi vähim ehk senini kõige halvem element (Algoritm 5.2). Huvitav on prioriteedijada puhul see, et kiirus paraneb oluliselt, kui ujukoma arve ümardada. Skooride ümardamisel kahe komakohani olulist informatsiooni ära ei kao, kuid kiirus paraneb kordi. Ajalised võrdlused 100 000 parima esinemise leidmise kohta on tootud tabelis 5.1.

Maatriks	***Aeg s (ümardamata)***	***Aeg s (ümardatud 2 kohani)***
M00014	333.6	118
M00197	627.3	122

Tabel 5.1

Maatrikseid on kahte tüüpi: algselt täisarvulised ning algselt reaalarvulised. M00014 on täisarvuline ning M00197 reaalarvuline. Nagu tabelist 3.1 näha, on ümardamisega saadud ajavõit eriti suur reaalarvulise maatriksi puhul. Põhjus peitub prioriteedijada ülesehituses. Ümardatud arvude puhul on erinevaid arve vähem ning seega sorteerimist prioriteedijada sees samuti vähem.

Algoritm 5.2 *n parima skooriga alamstringide leidmine*

Sisend: *Maatriks M[rows][columns], tekst T, parimate esinemiste arv n*

Väljund: *Struktuuride massiiv best_scores[n], kus salvestatakse skoor ning positsioon*

Meetod: *Kasutatakse meetodit* maxarray *(Algoritm 5.1). Kasutatakse prioriteedijada (best_scores). Esialgu* $\forall i$ *: best_scores[i].score = 0.*

```
1.  function Main
2.      double max_array = maxarray(M);
3.      match_1(M, T, l);
4. function match_1 (maatriks M, tekst T, n)
5.      double sum = 0.0;
6.      struct best best_scores;
7.      for (i = 0; i < length(l) – columns + 1; i++)
8.          for (j = 0; j < columns; j++)
9.              sum += M[T[i+j]][j]
10.             if (!(sum + max_array[j+1] > best_scores[1].score))
11.                 break
12.         best_scores[1].score = sum
13.         bubbleDown(best_scores, 1)
14.  return best_scores
15.function bubble_down (struct best best_scores[n], int i)
16.   if (best_scores[i].score > best_scores[2*i].score)
17.         swap(best_scores[i], best_scores[2*i]) //vaheta elemendid
18.         bubble_down(best_scores, 2*i)
19.   if (best_scores[i].score > best_scores[2*i+1].score)
20.         swap(best_scores[i], best_scores[2*i+ 1]) //vaheta elemendid
21.         bubble_down(best_scores, 2*i+1)
```

5.3.2 Kõikide heade esinemiste leidmine

Suurim probleem, mis kaalumaatriksi sobitamisega tekstile tekib, on õige lävendi valimine. Ühest head vastust sellele küsimusele pole siiani leitud. Siiski on ühes asjas suhteliselt üksmeelele jõutud: õige lävend peaks olema selline, et I liiki vea tekkimise tõenäosus oleks võimalikult väike ning ka II liiki vea tekkimise tõenäosus minimaalne. Probleem on aga selles, et isegi kui mõlemad tingimused teoreetiliselt kehtivad, on praktikas vigade arv suur. Oma otsingutes lähtusime sellest, et I liiki vea tõenäosus oleks võimalikult väike.

Kõige kindlam viis õige lävendi leidmiseks on kasutada jõumeetodit: võtta taustafail (pärmi puhul näit. pärmi kogu genoom), leida kõikide positsioonide skoorid, lugeda kokku skooride esinemissagedused ning selle põhjal leida skoor, millest suuremaid väärtusi leidub tõenäosusega 0.005. Teoorias on see küll lihtne, kuid praktikas väga aeglane.

Et programm kiiremaks muutuks, kasutatakse erinevaid statistilisi lähenemisi. Kõige esimene mõte oli kasutada taustafaili asemel maatriksit ennast. Kui eeldada, et taustafail on täiesti juhuslik, siis võiks maatriksi kasutamine töötada. Mõte on selles, et kui leida kõikvõimalikud skoorid, mis on võimalik maatriksiga saada, ning leida iga skoori tõenäosuse, saaks samuti leida lävendi, millest suurema skoori leidmise tõenäosus juhuslikult on 0.005. Selleks peab aga tõestama, et taustafailist leitud skooride jaotus on sama mis maatriksi abil leitud skooride jaotus. Nagu näha joonistelt 5.1, on silmajärgi jaotused tõesti väga sarnased.

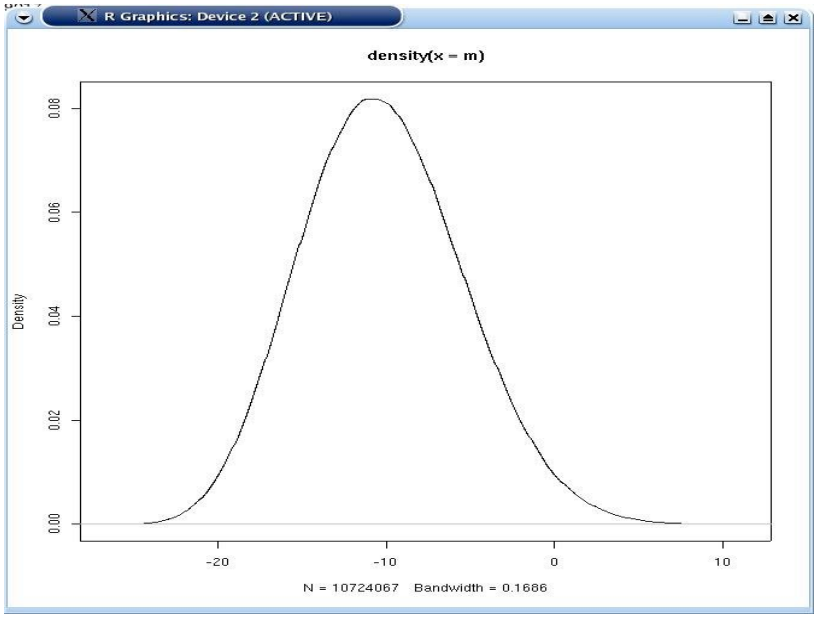
Püstitan hüpoteesi:

H0: Taustafaili skoorid ning maatriksi skoorid on sama jaotusega
H1: Taustafaili skoorid ning maatriksi skoorid ei ole sama jaotusega

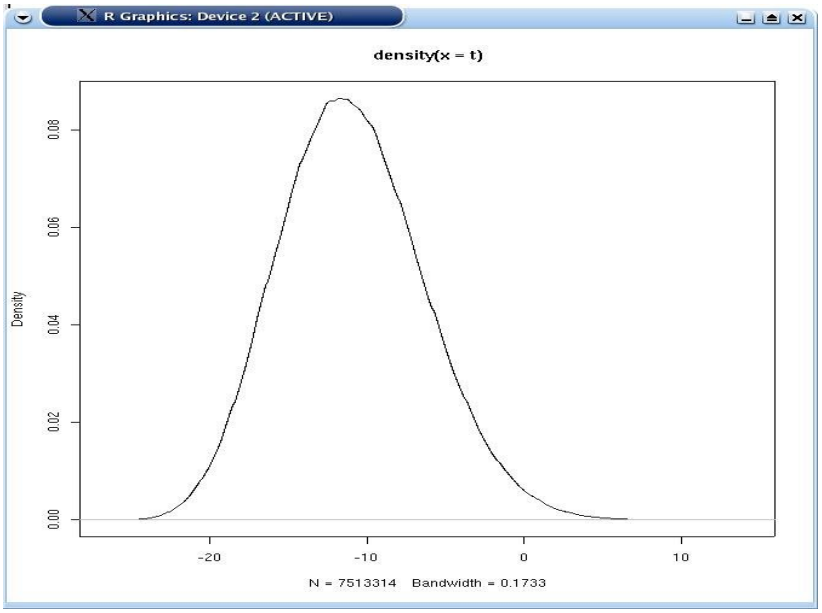
Hüpoteesi tõestamiseks leian kõigepealt hii-ruut statistiku. Selleks jagan kõigepealt võimalikud skooride väärtused sajaks grupiks. Nende põhjal arvutan hii-ruut statistiku. Statistiku väärtus on kõigi testitud maatriksite puhul tuhandetes, hii-ruut jaotuse 0.05 kvantiil 99 vabadusastme korral aga 123.23. Seega peame kindlasti vastu võtma H1, st taustafaili skoorid ning maatriksi skoorid ei ole sama jaotusega. See tähendab, et paraku ei saa otseselt kasutada maatriksit lävendi arvutamiseks.

Tänapäeval tegelevad mitmed uurimisgrupid lävendi leidmiseks sobivate statistiliste meetodite leidmisega, kuid ühtegi väga täpset meetodit pole veel leitud. Mõnede levinumate meetodite kohta võib lugeda allikatest [Bej03] ja [HS99]. Sama probleemi on proovitud lahendada ka juhuvaliku (ingl. k. *sampling*) abil [BE+04]. Lävendi arvutamise jaoks kiirema meetodi leidmine jääb üheks käesoleva töö edasiarenduse võimaluseks.

Joonis 5.1. Skooride jaotus taustafailis (a) ja maatriksis (b)



(a)



(b)

5.3.3 Maatriksi parandamine

Valminud tööriista üheks otstarbeks on ka maatriksi parandamine. Hea maatriks on selline, mille tekstile sobitades saavad parimad skoorid sellised alamjärjestused, mis bioloogiliselt olulised on. Sekventside bioloogilist tähendust saab kinnitada küll ainult *in vitro* katsete abil, kuid *in silico* on siiski võimalik teatud ennustusi teha. Maatriksi parandamise all peetakse silmas maatriksi muutmist nii, et tulemus mingi kriteeriumi järgi paremaks muutub. Näiteks võib arvata, et kui head esinemised koonduvad kindlasse piirkonda DNA-l, siis on tõenäosus, et leiti tõepoolest olulised alamsekventsid, suurem kui hittide suvalisel paiknemisel. Nagu eespoolgi mainitud, saab selliseid aspekte kindlaks teha vaid *in vitro*, kuid *in silico* saab vajalike eksperimentide arvu tunduvalt vähendada.

Maatriksi parandamise põhimõte seisneb selles, et sobitades maatriksit tekstile leitakse parimad esinemised ning moodustatakse nende põhjal uus maatriks. Sama protsessi korrates võib kokkuvõttes leida maatriksi, mis on eespool toodud mõistes parem kui esialgne.

Maatriksi headuse hindamine nõuab pikema-ajalist katsetamist. Seetõttu jääb see edasiseks arendusvõimaluseks.

6. Peatükk

Tulemused

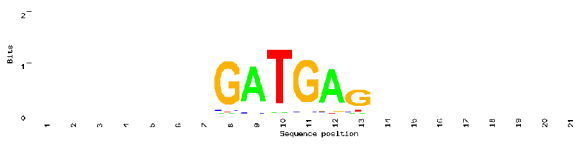
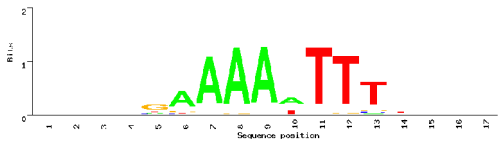
Bakalaureusetöö käigus valmis rakendus maatriksite sobitamiseks tekstile. Töö eesmärgiks oli parandada bioloogia seisukohast oluliste alamjärjestuste leidmist ning viia valepositiivsete ja valenegatiivsete tulemuste osakaal võimalikult väikeseks. SPEXS, mille edasiarendusena käesolev töö on mõeldud, kasutab mustrite otsimisel regulaaravaldisi. Elusloodusesmängivad aga rolli hoopis ainete keemilised omadused. Seetõttu ei pruugi ühe regulaaravaldise poolt kirjeldatud erinevad alamjärjestused olla samale transkriptsioonifaktorile seondumissaitideks.

Võttes kasutusele maatriksid samaks otstarbeks, lootsimegi parandada õigete seondumissaitide leidmist. Tabelit 6.1 vaadates võib näha, et tulemus tõesti paranes. Binoomtõenäosused (tõenäosus, et selline tulemus võib tekkida juhuslikult) maatriksite abil leitud saitide jaoks on tunduvalt väiksemad kui regulaaravaldiste abil leitud saitide jaoks. Tabel 6.1 on koostatud tööd läbivate näidete põhjal (Tabel 1.1, Näide 3.1, Näide 4.1).

Motiiv	Binoomtõenäosus regulaaravaldisega otsides	Binoomtõenäosus maatriksiga otsides
G.GATGAG.T	1.12e-37	9.03e-50
TG.AAA.TTT	1.84e-32	2.98e-51

Tabel 6.1

Motiivile vastavad kaalumaatriksid logo kujul olid järgmised:

G.GATGAG.T	
TG.AAA.TTT	

Kokkuvõttes valmis bakalaureusetöö jooksul rakendus maatriksite sobitamiseks tekstile. Programm on kirjutatud suuremas osas C-keeles, sisendi töötlemisega tegelevad klassid perlis. Tööriistale peab ette andma tekstifaili, maatriksid või ühepikkused sõned, millest maatriks genereeritakse, ning väljundfaili. Lisaks võib otsida *n* parimat esinemist tekstifailist kokku või *n* parimat igast tekstifaili järjestusest. Alati võib ka ette anda lävendi, millest väiksemaid tulemusi ei väljastata. Lähiajal on plaanis tööriistale ka veebiliides koostada.

Kokkuvõte

Käesoleva töö eesmärgiks oli arendada edasi tekstist mustrite otsimise tööriistu. Käesolev töö on otseselt seotud transkriptsiooni uuringutega, täpsemalt transkriptsioonifaktorite seondumissaitide leidmisega.

Praktilise töö moodustab tööriistakomplekt, mis on mõeldud ennekõike transkriptsioonifaktorite seondumissaitide asukohtade määramiseks, kuid on kasutatavad ka teistel eesmärkidel. Valminud tööriistakomplekt koosneb kolmest osast: klasterdamis-, joondamis- ja sobitamistööriistast. Kaks esimest valmisid autori semestritöö käigus, kolmas ja kõige mahukam käesoleva bakalaureusetöö käigus. Kõik kolm on ennekõike mõeldud sidumiseks olemasoleva tööriistaga SPEXS, kuid on ka eraldiseisvalt kasutatavad. Valminud programmide lähtekoodid asuvad aadressil http://kotkas.ebc.ee/u/tasa/.

Kuna praktiline töö on mõeldud opereerimiseks lühikeste järjestustega, seab see piirangud sisendile. Üks võimalus töö edasiarendamiseks on tööriista üldisemaks muutmine. See nõuab suuremaid muudatusi joondamise algoritmis ja mõningaid modifikatsioone nii klasterdamise kui sobitamise juures. Samuti tuleks uurida peidetud markovi ahelate kasutusvõimalusi, eriti klasterdamise juures. Lähiajal on plaanis kõigile kolmele tööriistale ka veebiliides programmeerida.

SUMMARY

Use of Position Weight Matrices in Bioinformatics

Bachelor Thesis

Triinu Tasa

Abstract

Today, weight matrices and their applications make up an important part of bioinformatics. Here we look into weight matrices' construction methods, their properties and some applications. As a practical work we developed a toolkit consisting of three programs, each of them utilizing weight matrices in a rather different way.

The main purpose of the toolkit is to complement an existing tool SPEXS for transcription factor binding site detection.

The first program concentrates on pattern clustering using weight matrices. The purpose of the program is to organize SPEXS output into clusters.

The second program concerns multiple alignment of short sequences. Here we propose a simple but efficient method of using a special kind of matrix, the product of an alignment and a probability matrix, as a reference point. It is needed for organizing clusters into matrices.

The third program deals with matching matrices over text. The purpose of the tool is to find subsequences most similar to a given matrix.

The programs are firstly meant to postprocess SPEXS output, the application for finding patterns in sequences. All three are also practicable as separate programs.

VIITED

[ALP00] A. Amir, M. Lewenstein, E. Porat. Faster Algorithms for String Matching with *k* Mismatches. *J. Algorithm*, vol. 50, 2004, p. 257-275.

[AU00] A. V. Aho, J. D. Ullman. Foundations of Computer Science. W. H. Freeman/Computer Science Press, New York 2000

[BE+04] Y. Barash, G. Elidan, T. Kaplan, N. Friedman. CIS: Compound Importance Sampling Method for Transcription Factor Binding Site p-value Estimation. *Bioinformatics* 21(5), 2004

[Bej03] G. Bejerano. Efficient Exact p-Value Computation and Applications to Biosequence Analysis. Proceedings of RECOMB 2003

[Edd98] S. R. Eddy. Profile Hidden Markov Models. *Bioinformatics* 14(9):755-63.

[EG01] W. J. Ewens, G. Grant. Statistical Methods in Bioinformatics. 2004

[Gus99] D. Gusfield. Algorithms on Strings, Trees, and Sequences. Cambridge University Press,1999

[HS99] G. Z. Hertz, G. D. Stormo. Idenfitying DNA and protein patterns with statistically significant alignments of multiple sequences. *Bioinformatics* 15(7), 1999

[Kar99] R. Karchin. Hidden Markov Models and Protein Sequence Analysis. *Intelligent Systems for Molecular Biology*, 1999

[Kih03] J. Kiho. Algoritmid ja andmestruktuurid. TÜ Kirjastus, 2003

[Par89] A.-M. Parring. Sissejuhatus matemaatilisse statistikasse. TRÜ trükikoda, Tartu, 1989

[Pea01] W. R. Pearson. Protein sequence comparison and Protein evolution: Tutorial. ISMB handouts 2001

[RMV03] S. Rahmann, T. Müller, M. Vingron. On the Power of Profiles for Transcription Factor Binding Site Detection. *Stat Appl Genet Mol Biol*,

2003

[SFW83] G. Salton, E. A. Fox, H. Wu: Extended Boolean Information Retrieval. Commun. ACM 26(11): 1022-1036 (1983)

[TM97] E.-M. Tiit, M. Möls. Rakendusstatistika algkursus. Tartu,1997

[Vil02] Jaak Vilo. Pattern Discovery from Biosequences. PhD Thesis, 2002

[WNB00] T. D. Wu, C. G. Nevill-Manning, D. L. Brutlag. Fast probabilistic analysis of sequence function using scoring matrices. *Bioinformatics* Vol. 16 no. 3, 2000

URL-id

[url:COL] Honig Lab at Columbia University. Positional Weight Matrix for Representation of Transcription Factor Binding Sites. (29.03.2005) <http://trantor.bioc.columbia.edu/search_for_BS/manual/matrix.html>

[url:EGK] T. Maimets. Mis on geen ja kuidas ta töötab? (29.03.2005) <http://www.genomics.ee/index.php?lang=est&show=23&sub=76>

[url:NTU] <http://bioinfo.csie.ntu.edu.tw/biolearn/Chinese/global%20alignment/global%20alignment/dynamic.programming-5.gif>

[url:SEQLOGO] <http://ep.ebi.ac.uk/EP/SEQLOGO/>

[url:SPEXS] <http://ep.ebi.ac.uk/EP/SPEXS/>