Sekvenssien segmentointi

ja dynaaminen ohjelmointi

- T-61.2010 Datasta tietoon
- Heikki Mannila, Jouni Seppänen syksy 2007

Sekvenssien segmentointi

- Erilaisia aikasarjoja esiintyy usein
- Miten aikasarja voidaan jakaa paloihin (segmentteihin) jotka ovat keskenään homogeenisia?
- Dynaamisen ohjelmoinnin algoritmi: voimakas työkalu

Erilaisia sekvenssejä

- Merkkijonot: $a_1 a_2 \cdots a_n$, $a_i \in \Sigma$
- Aikasarjat: $a_1, a_2, \ldots, a_n$, $a_i \in \mathbb{R}$
- Tapahtumasekvenssit:

$$((e_1, t_1), (e_2, t_2), \ldots, (e_n, t_n))$$

$e_i \in \Sigma$ tapahtumatyypit

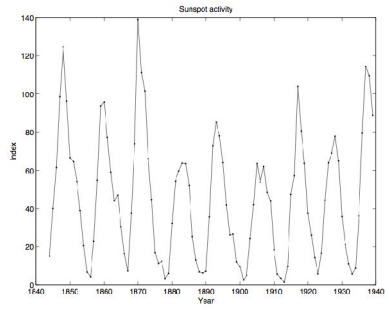
$t_i \in \mathbb{R}$ esiintymisajat

Merkkijonoja

- Dokumentit
- DNA

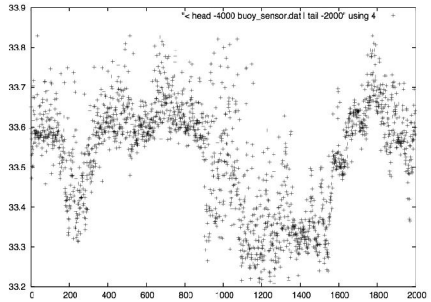
```
tataacacaaaataataaatgcttgaggggatgaatatccaattttccatt
atgtacttattgtacattgcatgcctgtaccaaaatatttcatgtacctc
ataaatgtatacacctgctatgtacccacaaaaattaaattttaaaacaa
tacattgttatccactatagtcaccatattgcacaatagatctgttgaat
tcattcctcctgtacaatgcaattttgtacccttgaccaacatctaccc
aatcctcctggtaaccatcattctactctgtacttctatgtgttcagcct
tcttagacctccacatacaagtgagattatgcagtatctggctttctgtg
cctggattattttactcagtataatgtcctcccggttcattcatgttgtc
acaaatgatactttttttatttttaaggttgtatactattctattgtgt
atgtgtaccacattttcttcatccactcatgtgtcgatggatacttaagt
taattccacatcttggctgttgtgaataatgctacaataaaatatgggagt
acagataactcattgacacactgatttgatatctttttaatatatgccca
```

Aikasarjoja



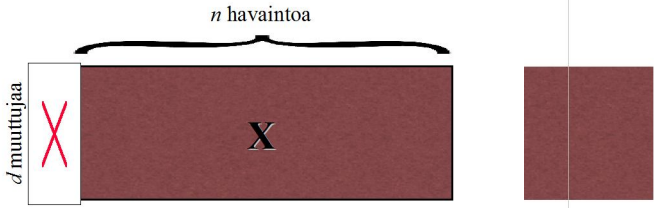
5

Aikasarjoja



6

Sekvenssin segmentointi

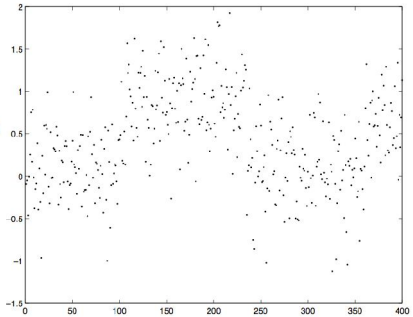


- Miten havaintojono voidaan jakaa osiin, joista kukin on sisäisesti samankaltainen?

7

Esimerkki

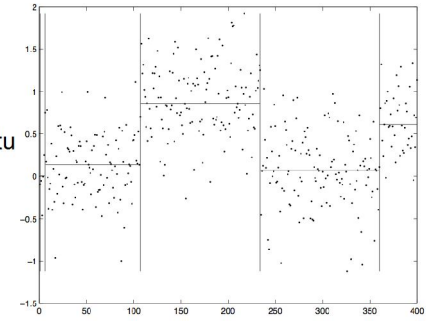
•Yksi-ulotteinen aikasarja, 400 pistettä



8

Esimerkki

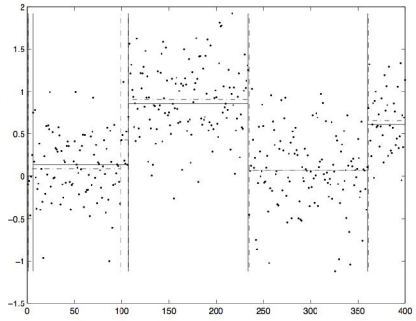
•Mistä data on generoitu



9

Esimerkki

•Mitä segmen-tointi löytää



10

Todellinen esimerkki

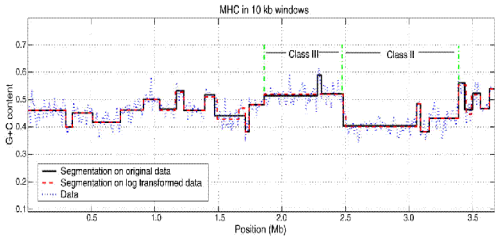


Fig. 4. G+C content of the human MHC region in 1 kb (upper plot) and 10 kb (lower plot) windows, segmentations for window sizes 1 kb and 10 kb, with and without log-transformation. Suggested boundaries obtained for the known class III and II isochores are shown.

Haiminen & Mannila, Discovering isochores by least-squares optimal segmentation, Gene 397 (2007), 53-60.

11

Merkintöjä

- Yksiulotteinen tilanne: $d = 1$
- Havaittu data: $y_1, y_2, y_3, \ldots, y_n$ (reaalilukuja)
- Segmentoidaan datajono k:hon segmenttiin
  - Segmentin i alkukohta b_i ja loppukohta e_j
  - (tällöin $b_1 = 1$, $e_j = b_{j+1} - 1$, $e_k = n$)
  - Toisin sanottuna segmentti i on

$$y_{b_j}, y_{b_{j+1}}, \ldots, y_{e_j}$$

12

Paloittain vakio esitys

- Esitetään kunkin segmentin j havaintoja vakiolla w_j
- Kuinka suuri virhe tehdään?
  - Katsantokannasta riippuen esim.

$$\sum_{j=1}^{k}\sum_{i=b_j}^{e_j}|y_i - w_j| \quad \text{tai} \quad \sum_{j=1}^{k}\sum_{i=b_j}^{e_j}(y_i - w_j)^2$$

Virheen minimointi

- Syöte:
  - data $y_1, y_2, y_3, \ldots, y_n$
  - segmenttien lukumäärä k
- Tuloste:
  - segmenttien alut b_j ja loput e_j
  - segmenttien vakiotasot w_j
- Tavoite: mahdollisimman pieni virhe

Virheen minimointi (2)

- Havainto 1: jos segmenttien rajat b_j, e_j tiedetään, tasot w_j on helppo löytää

$$\frac{\mathrm{d}}{\mathrm{d}w_j}\sum_{i=b_j}^{e_j}(y_i - w_j)^2 = \frac{\mathrm{d}}{\mathrm{d}w_j}\sum_i(y_i^2 - 2y_i w_j + w_j^2)$$

$$= \sum_i(2w_j - 2y_i) = 2w_j(e_j - b_j + 1) - 2\sum_i y_i$$

$$= 0 \iff w_j = \frac{\sum_{i=b_j}^{e_j} y_i}{e_j - b_j + 1}$$

Virheen minimointi (3)

- Havainto 2: mahdollisuuksia segmenttien rajoiksi on paljon
  - esim. 400 pistettä, 4 segmenttiä:

$$\binom{400}{4} \approx \frac{400^4}{4!} \approx 10^9 \text{ vaihtoehtoa}$$

- ”Kokeillaan kaikki vaihtoehdot” vaikuttaa huonolta idealta
  - mutta joskus huonoakin ideaa voi korjata...

Fibonaccin luvut

- Varhaista populaatiogenetiikkaa (1202): ajanhetkellä $n + 2$ ovat elossa ajanhetken $n + 1$ kanit ja (nyt sukukypsien) ajanhetken n kanien jälkeläiset
  - $F_1 = F_2 = 1$
  - $F_{n+2} = F_n + F_{n+1}$

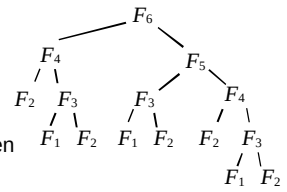
Fibonaccin luvut (2)

- Huono idea Fibonaccin lukujen laskemiseksi:
  - käytetään määritelmää $F_{n+2} = F_n + F_{n+1}$ rekursiivisesti

$$\begin{aligned} F_6 &= F_4 + F_5 \\ &= (F_2 + F_3) + F_5 \\ &= 1 + (F_1 + F_2) + F_5 \\ &= 1 + 1 + 1 + F_3 + F_4 \\ &= 3 + (F_1 + F_2) + F_4 \\ &= 3 + 1 + 1 + (F_2 + F_3) \\ &= 5 + 1 + (F_1 + F_2) \\ &= 6 + 1 + 1 = 8 \end{aligned}$$

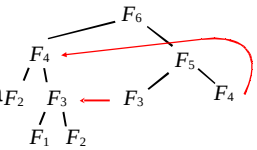
Fibonaccin luvut (3)

- Miksi idea on niin huono?
  - Sama asia lasketaan moneen kertaan



Fibonaccin luvut (4)

- Parempi idea: käytetään määritelmää rekursiivisesti mutta muistetaan mitä on jo laskettu
- Mitä tekemistä tällä on segmentoinnin kanssa?



Virheen minimointi (4)

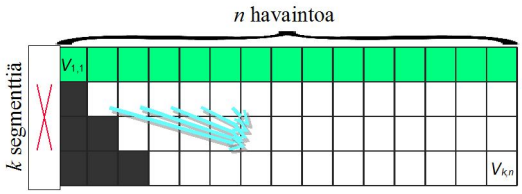
- Syöte: data $y_1, y_2, y_3, \ldots, y_n$; segm. määrä k
- Tuloste: segm. alut ja loput ja vakiotasot
- Tavoite: mahdollisimman pieni virhe
- Havainto 3: virhe on segmenttikohtainen:
  - Oletetaan optimaalinen segmentointi
  - Poistetaan datasta viimeinen segmentti
  - Alkuosan segmentointi on edelleen optimaalinen!

Virheen minimointi (5)

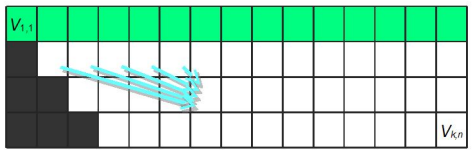
- Havainto 3: virhe on segmenttikohtainen:
  - Oletetaan optimaalinen segmentointi
  - Poistetaan datasta viimeinen segmentti
  - Alkuosan segmentointi on edelleen optimaalinen!
- Miksi?
  - Jos alkuosalle olisi parempi ratkaisu, sitä voisi käyttää koko datan segmentointiin.

Dynaaminen ohjelmointi

- Tehdään taulukko V kokoa $k \times n$, jonka alkio $V_{j,i}$ kertoo pienimmän mahdollisen virheen, kun datajono $y_1, \ldots, y_i$ jaetaan j segmenttiin.



Dynaaminen ohjelmointi (2)



- $V_{1,i}$ on helppo laskea (havainto 1)
- Havainto 4: $V_{j,i}$ on laskettavissa luvuista $V_{j-1,i'}$ $(i' < i)$

Dynaaminen ohjelmointi (3)

- Havainto 4: $V_{j,i}$ on laskettavissa luvuista $V_{j-1,i'}$ ($i' < i$).
  - Paras i:n pisteen j-segmentointi saadaan lisäämällä yksi segmentti parhaaseen i':n pisteen ($j-1$)-segmentointiin jollakin $i' < i$.
  - Seuraa havainnosta 3.

Virheen minimointi (6)

- Ohjelman runko:
  - Lasketaan kaikkien mahdollisten segmenttien kustannukset $K_{p,q}$.
  - Alustetaan dynaamisen ohjelmoinnin taulukko: $V_{1,i} = K_{1,i}$.
  - Lasketaan taulukon jokainen rivi $V_{j,*}$ vasemmalta oikealle.
  - Rekonstruoidaan segmentointi aputaulukosta P.

Ohjelmarunko koodina

```
function segm = seg(data, nsegments)

K = segment_costs(data);
V = init_array(K, nsegments);
[V, P] = dynprog(K, V);
segm = reconstruct(P, data, K);
fprintf(1, 'Minimum cost is %f\n', V(end, end));
fprintf(1, ['Achievable by the following segmentation ', ...
            '(beg, end, constant, cost):\n']);
disp(segm);
```


1. Segmenttikustannukset

- Jos datajonon $D = (y_1, \ldots, y_n)$ osajono $D(p{:}q)$ tulee käytetyksi segmenttinä eli jos $b_j = p$ ja $e_j = q$ jollakin j, niin segmentin vakioksi tulee

$$w_j = \frac{y_p + y_{p+1} + \cdots + y_q}{q - p + 1}$$

- Tälle segmentille kohdistuva virhe on

$$K_{p,q} = \sum_{i=p}^{q} (y_i - w_j)^2$$

Koodina

```
function K = segment_costs(data)

n = length(data);
K = inf * ones(n, n);
for p = 1:n,
  for q = p:n,
    w = mean(data(p:q));
    K(p,q) = sum((data(p:q) - w) .^ 2);
  end
end
```


2. Taulukon alustus

- Taulukon *V* ensimmäiselle riville $V_{1,*}$ tulevat 1-segmentointien kustannukset.
- Ne itse asiassa laskettiin jo taulukkoon *K*:
  - $V_{1,i} = K_{1,i}$.
- (Taulukon *K* muita rivejä käytetään myöhemmin.)

Koodina

```
function V = init_array(K, nsegments)

n = size(K, 2);
V = zeros(nsegments, n);
for i = 1:n,
  V(1,i) = K(1,i);
end
```

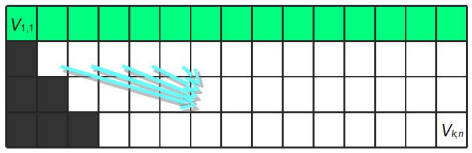

3. Optimointi

- Havainnon 4 avulla täytetään nyt taulukko rivi kerrallaan:

$$V_{j,i} = \min_{i' < i} \{ V_{j-1,i'} + K_{i'+1,i} \}$$

Alkupään virhe

Viimeisen segmentin virhe



Koodina

```
function [V, P] = dynprog(K, V)
[nsegments, n] = size(V);
P = zeros(nsegments, n);
for j=2:nsegments,
  for i=j:n,
    best = inf; idx = -1;
    for ii=1:i-1,
      cost = V(j-1, ii) + K(ii+1, i);
      if cost < best, best = cost; idx = ii; end
    end
    V(j,i) = best; P(j,i) = idx;
  end
end
```


4. Segmentointi

- Edellinen ohjelma laskee parhaan segmentaation virheen mutta ei itse segmentaatiota.
- Muutetaan ohjelmaa hieman: taulukon *V* lisäksi pidetään yllä taulukkoa *P*, jonka alkio *i'* = $P_{j,i}$ kertoo, mikä $P_{j-1,i'}$ havaittiin pienimmäksi $V_{j,i}$:tä laskettaessa.
- Taulukkoa *P* seuraamalla löydetään optimaalinen segmentaatio.

Koodina

```
function segm = reconstruct(P, data, K)

[nsegments, n] = size(P);
segm = zeros(nsegments, 4);
i = n;
for j=nsegments:-1:1,
  start = P(j,i) + 1;
  segm(j,:) = [start, i, ...
               mean(data(start:i)), ...
               K(start,i)];
  i = P(j,i);
end
```


Aikavaatimus

- Tietojenkäsittelytieteen fundamentaalisin kysymys:
  - Mitä voidaan laskea mekaanisesti?
  - (Vastaus: kaikki erilaiset mekaanisen laskennan formulaatiot ovat osoittautuneet yhtä voimakkaiksi.)
- Seuraava kysymys:
  - Mitä voidaan laskea *tehokkaasti?*

Aikavaatimus (2)

- Suoraviivaisin tapa mitata ajoaikaa: lasketaan montako alkeisoperaatiota prosessori tekee. Ongelmia:
  - prosessorien erot
  - kääntäjien erot
  - koodaajien erot
  - ja tekniikka kehittyy koko ajan

37

Aikavaatimus (3)

- Käytännössä voidaan laskea, montako sellaista ”alkeisoperaatioita” kuin kokonaislukujen yhteen- tai kertolasku suoritetaan
  - (Tarkalleen ottaen näidenkin suoritusaika riippuu lukujen suuruusluokasta.)

38

Aikavaatimus (4)

- Tarkoista tehokkuuslaskuista tulee alkeellisissakin tapauksissa hyvin monimutkaisia, joten melkein aina tyydytään tarkastelemaan asymptotiikkaa.
- Esimerkiksi funktio $3n^3 - n^2 + n + 100$ kasvaa suunnilleen samoin kuin n^3: vakiokerroin 3 ja alemman asteen termit eivät vaikuta. Kirjoitetaan

$$3n^3 - n^2 + 100 = O(n^3).$$

39

Aikavaatimus (5)

- Tarkemmin sanottuna

$$f(n) = O(g(n)) \text{ jos } \limsup_{n \to \infty} \frac{f(n)}{g(n)} < \infty.$$

- (Vastaavasti $f(n) = \Omega(g(n))$ jos $g(n) = O(f(n))$ ja

  $f(n) = \Theta(g(n))$ jos sekä $f(n) = O(g(n))$ että $f(n) = \Omega(g(n))$.)

40

Aikavaatimus (6)

- O(n), lineaarinen aika: Vakioaika jokaista data-alkiota kohden. Tämä aika kuluu jo datan lukemiseen.
- O(n log n): Lukujen suuruusjärjestykseen lajittelun viemä aika.
- O(n^2): Neliöllinen aika. Melko hyvä.
- O(n^3): Kuutiollinen aika. Alkaa olla hankala.
- O(2^n): Eksponentiaalinen aika, erittäin hidas.

Aikavaatimus (7)

- Sisimmän silmukan sisällä $O(n)+O(n)=O(n)$ operaatiota, silmukka suoritetaan $O(n)$ kertaa; siis $O(n^2)$ operaatiota

```
function K = segment_costs(data)

n = length(data);
K = inf * ones(n, n);
for p = 1:n,
  for q = p:n,
    w = mean(data(p:q));
    K(p,q) = sum((data(p:q) - w) .^ 2);
  end
end
```

n = O(n)

q–p = O(n)

n–p = O(n)

n = O(n)

- Ulompi silmukka suoritetaan myös $O(n)$ kertaa; siis kaikkiaan $O(n^3)$ operaatiota.

Aikavaatimus (8)

```
function V = init_array(K, nsegments)

n = size(K, 2);
V = zeros(nsegments, n);
for i = 1:n,
  V(1,i) = K(1,i);
end
```

kn = O(kn)

n = O(n)

O(1)

- $O(kn)$ operaatiota.

Aikavaatimus (9)

O(kn)

O(k)

O(n)

O(n)

```
function [V, P] = dynprog(K, V)
[nsegments, n] = size(V);
P = zeros(nsegments, n);
for j=2:nsegments,
  for i=j:n,
    best = inf; idx = -1;
    for ii=1:i-1,
      cost = V(j-1, ii) + K(ii+1, i);
      if cost < best, best = cost; idx = ii; end
    end
    V(j,i) = best; P(j,i) = idx;
  end
end
```

- $O(kn+kn^2) = O(kn^2)$ operaatiota.

Aikavaatimus (10)

```
function segm = reconstruct(P, data, K)

[nsegments, n] = size(P);
segm = zeros(nsegments, 4);
i = n;
for j=nsegments:-1:1,
  start = P(j,i) + 1;
  segm(j,:) = [start, i, ...
          mean(data(start:i)), ...
               K(start,i)];
  i = P(j,i);
end
```

O(k) — O(n)

- $O(kn)$ operaatiota.

Aikavaatimus (11)

```
K = segment_costs(data);
V = init_array(K, nsegments);
[V, P] = dynprog(K, V);
segm = reconstruct(P, data, K);
```

$O(n^3)$ $O(kn)$ $O(kn^2)$ $O(kn)$

- $O(n^3+kn+kn^2+kn) = O(n^3+kn^2)$ operaatiota
- Oletettavasti $k < n$, joten termi n^3 hallitsee
- Voisiko kriittistä kohtaa nopeuttaa?

Segmenttikustannukset

- Katsotaan tarkemmin segmenttikohtaista virhettä:

$$\sum_{i=p}^{q}(y_i - w)^2 = \sum_i (y_i^2 + w^2 - 2y_i w)$$
$$= \sum_i y_i^2 + (q-p+1)w^2 - 2w\sum_i y_i$$
$$= \sum_i y_i^2 + (q-p+1)w^2 - 2w(q-p+1)w$$
$$= \sum_i y_i^2 - (q-p+1)w^2 = \sum_i y_i^2 - \frac{(\sum_i y_i)^2}{q-p+1}$$

Nopeampi `segment_costs`

$$\sum_{i=p}^{q}(y_i - w)^2 = \sum_i y_i^2 - \frac{(\sum_i y_i)^2}{q-p+1}$$

```
function K = segment_costs(data)
n = length(data);
K = inf * ones(n, n);
for p = 1:n,
  data_sum = 0; % data_sum = data(p) + ... + data(q)
  sq_sum = 0;   % sq_sum = data(p)^2 + ... + data(q)^2
  for q = p:n,
    data_sum = data_sum + data(q);
    sq_sum = sq_sum + data(q)^2;
    K(p,q) = sq_sum - data_sum^2 / (q-p+1);
  end
end
```

O(n) — O(n)

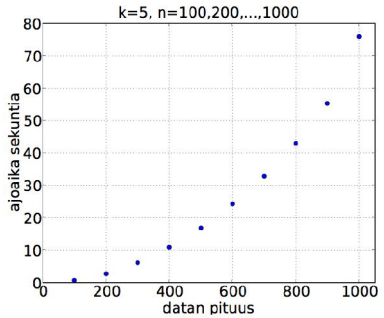
- $O(n^2)$ operaatiota.

Aikavaatimus (12)

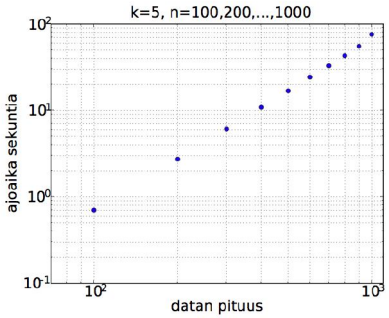
$O(n^2)$	`K = segment_costs(data);`
$O(kn)$	`V = init_array(K, nsegments);`
$O(kn^2)$	`[V, P] = dynprog(K, V);`
$O(kn)$	`segm = reconstruct(P, data, K);`

- $O(n^2+kn+kn^2+kn) = O(kn^2)$ operaatiota

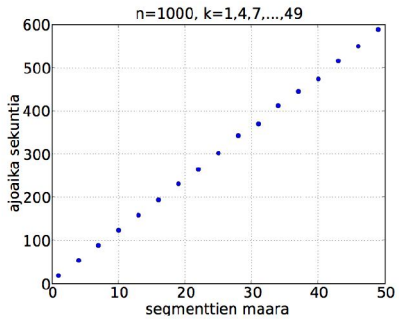
Todellinen ajoaika



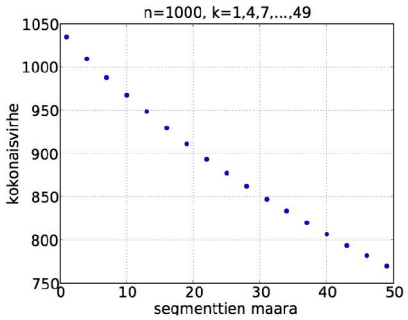
Todellinen ajoaika (2)



Todellinen ajoaika (3)



Virhe



Muunnelmia

- *d*-ulotteinen data, $d > 1$
- Eri virhemittoja:
  - L_1 (virheiden summa)
  - L_2 (tässä käytetty)
  - L_p
  - L_∞ (virheiden maksimi)

Eri muuttujien perusteella tehtyjä segmentaatioita

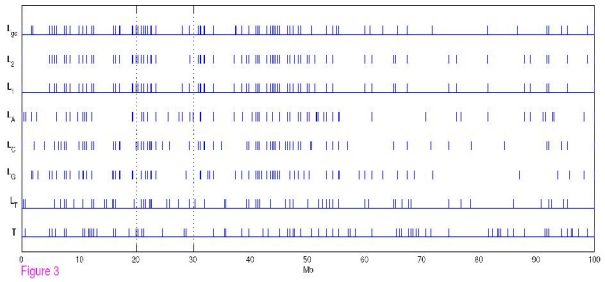


Figure 3

Niina Haiminen, Heikki Mannila, Evimaria Terzi: Comparing segmentations, *BMC Bioinformatics 2007*

Muunnelmia (2)

- Tapoja approksimoida dataa
  - paloittain vakio (tässä käytetty)
  - paloittain lineaarinen
  - paloittain neliöllinen
- Kaikki nämä onnistuvat muuttamalla taulukon *K* laskentaa.

Muunnelmia (3)

- Esimerkki vaikeammasta muunnelmasta:
  - paloittain vakio esitys
  - k segmenttiä, virhemittana L_1 tai L_2
  - segmenttien vakiotasoina saa käyttää vain h erisuurta lukua ($h < k$)
- Havainto 3 ei pädekään: jos poistetaan viimeinen segmentti, aiempia voi kannattaa muuttaa

- Gionis, A. & Mannila, H. **Finding Recurrent Sources in Sequences**. *RECOMB* 2003.

57

Ongelmia

- ”Oikean” k:n valinta
  - Millä tahansa k:lla saadaan jokin ratkaisu.
  - Erilaiset korjaustermit.
- Algoritmin nopeuttaminen
  - $O(kn^2)$ on turhan hidas suurilla n
  - Online-ongelmat: dataa tulee koko ajan ja sitä pitäisi segmentoida saman tien.

58

Dynaaminen ohjelmointi

- Dynaaminen ohjelmointi sopii optimointitehtäviin, joissa
  - optimaalinen ratkaisu sisältää optimaalisia ratkaisuja osatehtäviin
  - samoja osatehtäviä joudutaan ratkomaan toistuvasti.

59

Dynaaminen ohjelmointi

- Matriisitulon

$$M_1 M_2 \cdots M_n$$

paras laskujärjestys, kun matriisien koot vaihtelevat.

- Idea: viimeisenä lasketaan *jokin* tulo

$$(M_1 \cdots M_k)(M_{k+1} \cdots M_n)$$

jonka osatulot ovat optimaalisia.

60

Dynaaminen ohjelmointi

- Merkkijonojen (esim. DNA:n) editointietäisyys: mikä on pienin määrä yhden merkin poistoja ja lisäyksiä, joilla GGGAATTGCA muutetaan GGAATTTCGA:ksi?
  - GGGAATT*GCA
  - GG*AATTTCGA
- Needlemanin–Wunschin algoritmi (1970)

61