Todistus. Katso Chang & Lampe [3]. □

- Algoritmi 2.7.6 toimii hyvin käytännössä, jos k on pieni.
- Algoritmi 2.7.6 toimii myös muille etäisyysmitoille, joilla lävistäjät ovat monotonisia (arvot eivät koskaan laske), esim. (poisto/lisäys)-editointietäisyys.
- Chang & Lampe [3] esittävät toisenlaisen algoritmin, joka toimii keskimäärin ajassa $O(kn)$.
- Lävistäjälemmaan perustuen on kehitetty useita likimääräisen hahmonsovituksen algoritmeja, joiden aikavaatimus on $O(kn)$ pahimmassakin tapauksessa (eikä vain keskimäärin), esim. Landau & Vishkin [9].

2.7.1 Myersin bittirinnakkainen algoritmi

G. Myers. A fast bit-vector algorithm for approximate pattern matching based on dynamic programming. *JACM* 46(3):395–415, 1999.

- Nopeutetaan matriisin (g_{ij}) laskentaa soveltamalla bittirinnakkaisuutta. Rajoitumme lyhyisiin hahmoihin, joilla matriisin sarake voidaan käsitellä yhdessä sanassa.
- Matriisi (g_{ij}) esitetään muodossa, jossa talletetaan solujen arvojen sijasta kahden vierekkäisen solun erotus.
- Määritellään *vaakaero* (horizontal delta) $\Delta h_{ij} = g_{ij} - g_{i,j-1}$ ja *pystyero* (vertical delta) $\Delta v_{ij} = g_{ij} - g_{i-1,j}$.
- Koska $g_{i0} = i$ ja $g_{0j} = 0$, pätee

$$\begin{aligned} g_{ij} &= \Delta v_{1j} + \Delta v_{2j} + \cdots + \Delta v_{ij} \\ &= i + \Delta h_{1j} + \Delta h_{2j} + \cdots + \Delta h_{ij} \end{aligned}$$

- Kaikki arvot Δh_{ij} ja Δv_{ij} voidaan esittää kahdella bitillä, mikä seuraa Lemmasta 2.7.8.

Lemma 2.7.8 *Kaikilla i, j on voimassa $\Delta h_{ij}, \Delta v_{ij} \in \{-1, 0, 1\}$.*

- Laskenta etenee sarakkeittain

$$\Delta v_{*0} \to \Delta h_{*1} \to \Delta v_{*1} \to \Delta h_{*2} \to \cdots$$

- Lisäksi lasketaan viimeisen rivin arvot $sc_j = g_{mj}$, $0 \leq j \leq n$, kaavoilla $sc_0 = m$ ja $sc_j = sc_{j-1} + \Delta h_{mj}$.

Esimerkki 2.7.9 *Vaaka- ja pystyerot hahmolle* `match` *ja tekstille* `remachine` *(-1 ja 1 on korvattu symboleilla $-$ ja $+$).*

		r		e		m		a		c		h		i		n		e	
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
m	+		+		+		0		+		+		+		+		+		+
	1	0	1	0	1	−	0	+	1	0	1	0	1	0	1	0	1	0	1
a	+		+		+		+		−		0		+		+		+		+
	2	0	2	0	2	−	1	−	0	+	1	+	2	0	2	0	2	0	2
t	+		+		+		+		+		0		0		+		+		+
	3	0	3	0	3	−	2	−	1	0	1	+	2	+	3	0	3	0	3
c	+		+		+		+		+		0		0		0		+		+
	4	0	4	0	4	−	3	−	2	−	1	+	2	+	3	+	4	0	4
h	+		+		+		+		+		+		−		−		−		0
	5	0	5	0	5	−	4	−	3	−	2	−	1	+	2	+	3	+	4

- Dynaamisen ohjelmoinnin matriisin solun g_{ij} laskemiseen tarvitaan soluja $g_{i-1,j}, g_{i-1,j-1}$ ja $g_{i,j-1}$. Näihin liittyy kaksi pystyeroa $\Delta v^{\text{in}} = \Delta v_{i,j-1}$ ja $\Delta v^{\text{out}} = \Delta v_{i,j}$ sekä kaksi vaakaeroa $\Delta h^{\text{in}} = \Delta h_{i,j-1}$ ja $\Delta h^{\text{out}} = \Delta h_{i,j}$.

- Todistuksessa (mutta ei laskennassa) käytetään lisäksi arvoa $\Delta d = g_{ij} - g_{i-1,j-1}$.

$$\begin{tikzcd}
g_{i-1,j-1} \arrow[r, "\Delta h^{\text{in}}"] \arrow[d, "\Delta v^{\text{in}}"'] \arrow[dr, "\Delta d"] & g_{i-1,j} \arrow[d, "\Delta v^{\text{out}}"] \\
g_{i,j-1} \arrow[r, "\Delta h^{\text{out}}"'] & g_{ij}
\end{tikzcd}$$

- Lisäksi määritellään, että $Eq = Eq_{ij}$ on 1, jos $p_i = s_j$, muutoin 0.
- Kuvataan laskenta funktiona, jonka argumentteina ovat Δv^{in}, Δh^{in} ja Eq sekä tuloksina Δv^{out} ja Δh^{out}.

Lemma 2.7.10 *Jos* $Eq = 1$ *tai* $\Delta v^{\text{in}} = -1$ *tai* $\Delta h^{\text{in}} = -1$,
niin pätee $\Delta v^{\text{out}} = -\Delta h^{\text{in}}$ *ja* $\Delta h^{\text{out}} = -\Delta v^{\text{in}}$.
Muuten $\Delta v^{\text{out}} = 1 - \Delta h^{\text{in}}$ *ja* $\Delta h^{\text{out}} = 1 - \Delta v^{\text{in}}$.

Todistus.

- g_{ij}:n palautuskaava voidaan kirjoittaa muotoon:

$$\begin{aligned} g_{ij} &= \min\{g_{i-1,j-1} + (\textbf{if } p_i = s_j \textbf{ then } 0 \textbf{ else } 1), g_{i,j-1} + 1, g_{i-1,j} + 1\} \\ &= g_{i-1,j-1} + \min\{1 - Eq, \Delta v^{\text{in}} + 1, \Delta h^{\text{in}} + 1\}. \end{aligned}$$

- Merkitään $\Delta d = g_{ij} - g_{i-1,j-1} = \min\{1 - Eq, \Delta v^{\text{in}} + 1, \Delta h^{\text{in}} + 1\}$. Siis $\Delta d = 0$, jos $Eq = 1$ tai $\Delta v^{\text{in}} = -1$ tai $\Delta h^{\text{in}} = -1$. Muuten $\Delta d = 1$.
- Selvästi $\Delta d = \Delta v^{\text{in}} + \Delta h^{\text{out}} = \Delta h^{\text{in}} + \Delta v^{\text{out}}$, mistä saadaan $\Delta v^{\text{out}} = \Delta d - \Delta h^{\text{in}}$ ja $\Delta h^{\text{out}} = \Delta d - \Delta v^{\text{in}}$.

□

Arvot Eq_{ij} ja Δd ovat bittejä mutta Δv_{ij} ja Δh_{ij} ovat "trittejä". Siirrytään nyt laskemaan kokonaan biteillä. Samalla siirrytään aritmeettisista loogisiin operaatioihin tulkinnalla 0 on epätosi ja 1 on tosi.

- Esitetään Δv_{ij} kahdella bitillä Pv_{ij} ja Mv_{ij}, joista edellisessä on 1, jos $\Delta v_{ij} = 1$, ja vastaavasti jälkimmäinen on 1, jos $\Delta v_{ij} = -1$. Esitetään Δh_{ij} vastaavasti kahdella bitillä Ph_{ij} ja Mh_{ij}. Tällöin

$$\begin{aligned} \Delta v_{ij} &= Pv_{ij} - Mv_{ij} \\ \Delta h_{ij} &= Ph_{ij} - Mh_{ij} \end{aligned}$$

Lemma 2.7.11

$$\begin{aligned} Pv^{\text{out}} &= Mh^{\text{in}} \vee \neg(Xv \vee Ph^{\text{in}}) \\ Mv^{\text{out}} &= Ph^{\text{in}} \wedge Xv \\ Ph^{\text{out}} &= Mv^{\text{in}} \vee \neg(Xh \vee Pv^{\text{in}}) \\ Mh^{\text{out}} &= Pv^{\text{in}} \wedge Xh \end{aligned}$$

missä $Xv = Eq \vee Mv^{\text{in}}$ *ja* $Xh = Eq \vee Mh^{\text{in}}$.

Todistus. Symmetrian vuoksi riittää osoittaa väite Pv:lle ja Mv:lle.

- Koska $\Delta d = \neg(Eq \vee Mv^{\text{in}} \vee Mh^{\text{in}})$, lemman 2.7.10 mukaan

$$\begin{aligned} Pv^{\text{out}} &= (\neg\Delta d \wedge Mh^{\text{in}}) \vee (\Delta d \wedge \neg Ph^{\text{in}}) \\ Mv^{\text{out}} &= (\neg\Delta d \wedge Ph^{\text{in}}) \vee (\Delta d \wedge 0) = \neg\Delta d \wedge Ph^{\text{in}} \end{aligned}$$

- Koska $\Delta d = \neg(Xv \vee Mh^{\text{in}}) = \neg Xv \wedge \neg Mh^{\text{in}}$, pätee

$$\begin{aligned} Pv^{\text{out}} &= ((Xv \vee Mh^{\text{in}}) \wedge Mh^{\text{in}}) \vee (\neg Xv \wedge \neg Mh^{\text{in}} \wedge \neg Ph^{\text{in}}) \\ &= Mh^{\text{in}} \vee \neg(Xv \vee Mh^{\text{in}} \vee Ph^{\text{in}}) \\ &= Mh^{\text{in}} \vee \neg(Xv \vee Ph^{\text{in}}) \\ Mv^{\text{out}} &= (Xv \vee Mh^{\text{in}}) \wedge Ph^{\text{in}} \\ &= Xv \wedge Ph^{\text{in}} \end{aligned}$$

□

Lemman perusteella koko matriisin bittiesitys voidaan laskea seuraavalla algoritmilla:

for $i := 1$ **to** m **do**
    $Pv_{i0} := 1$; $Mv_{i0} := 0$
for $j := 1$ **to** n **do**
    $Ph_{0j} := 0$; $Mh_{0j} := 0$
    for $i := 1$ **to** m **do**
        $Xh_{ij} := Eq_{ij} \vee Mh_{i-1,j}$
        $Ph_{ij} := Mv_{i,j-1} \vee \neg(Xh_{ij} \vee Pv_{i,j-1})$
        $Mh_{ij} := Pv_{i,j-1} \wedge Xh_{ij}$
    for $i := 1$ **to** m **do**
        $Xv_{ij} := Eq_{ij} \vee Mv_{i,j-1}$
        $Pv_{ij} := Mh_{i-1,j} \vee \neg(Xv_{ij} \vee Ph_{i-1,j})$
        $Mv_{ij} := Ph_{i-1,j} \wedge Xv_{ij}$

Tämä algoritmi ei ole vielä aiempaa tehokkaampi, mutta sen sisäsilmukat voidaan bittirinnakkaistaa.

- Talletetaan sarakkeet Pv_{*j}, Mv_{*j}, Ph_{*j}, Mh_{*j} ja Eq_{*j} kukin yhteen bittivektoriin. Oletetaan, että $m \leq w$ ja vektoreissa on bitit positioissa $1, \ldots, m$.[1]

[1] Algoritmissa viitataan Ph- ja Mh-vektoreiden 0-bittiin, mutta niiden arvo on aina 0, mikä tulee siirto-operaatiossa oikein huomioonotetuksi.

- Jälkimmäinen sisäsilmukka voidaan nyt suorittaa seuraavasti:

$$\begin{aligned}&Xv_{*j} := Eq_{*j} \vee Mv_{*,j-1}\\&Pv_{*j} := (Mh_{*,j} << 1) \vee \neg(Xv_{*j} \vee \ (Ph_{*j} << 1))\\&Mv_{*j} := (Ph_{*j} << 1) \wedge Xv_{*j}\end{aligned}$$

  Tässä on käytetty siirto-operaatiota $B << 1$, joka siirtää bittivektorissa B bitin kohdasta $i-1$ kohtaan i kaikilla $1 < i \leq |B|$ (ja asettaa $B_1 = 0$).

- Ensimmäisen silmukan bittirinnakkaistus aiheuttaa kuitenkin ongelman:

$$\begin{aligned}&Xh_{*j} := Eq_{*j} \vee (Mh_{*j} << 1)\\&Ph_{*j} := Mv_{*,j-1} \vee \neg(Xh_{*j} \vee \ Pv_{*,j-1})\\&Mh_{*j} := Pv_{*,j-1} \wedge Xh_{*j}\end{aligned}$$

  Nyt vektoria Mh_{*j} käytetään Xh_{*j}:n laskemiseen ennen kuin se on laskettu. Laskentajärjestyksen muutos ei auta, koska Xh_{*j}:tä käytetään puolestaan Mh_{*j}:n laskemiseen.

- Kehästä päästään ulos laskemalla Xh_{*j} ilman Mh_{*j}:tä tai muita vasta myöhemmin arvonsa saavia bittivektoreita. Seuraava lemma määrittelee Xh_{*j}:n vektoreiden Eq_{*j} ja $Pv_{*,j-1}$ avulla.

Lemma 2.7.12

$$Xh_{ij} = \exists \ell \in [1, i] : Eq_{\ell j} \wedge (\forall x \in [\ell, i-1] : Pv_{x,j-1}).$$

Todistus. Todistetaan väite induktiolla i:n suhteen.

- Lemman 2.7.11 mukaan

$$\begin{aligned}Xh_{ij} &= Eq_{ij} \vee Mh_{i-1,j}\\Mh_{ij} &= Pv_{i,j-1} \wedge Xh_{ij}\end{aligned}$$

- Alustus $i = 1$: Oikea puoli kutistuu muotoon Eq_{1j}. Koska $Mh_{0j} = 0$ kaikilla j, pätee $Xh_{1j} = Eq_{1j} \vee Mh_{0j} = Eq_{1j}$.

- Induktioaskel: Oletetaan, että $Xh_{i-1,j}$ on väitteen mukainen.

$$\begin{aligned} Xh_{ij} &= Eq_{ij} \vee Mh_{i-1,j} \\ &= Eq_{ij} \vee (Pv_{i-1,j-1} \wedge Xh_{i-1,j}) \end{aligned}$$

  Toisaalta

$$\begin{aligned} &\exists \ell \in [1,i] : Eq_{\ell j} \wedge (\forall x \in [\ell, i-1] : Pv_{x,j-1}) \\ &= Eq_{ij} \vee \exists \ell \in [1,i-1] : Eq_{\ell j} \wedge (\forall x \in [\ell, i-1] : Pv_{x,j-1}) \\ &= Eq_{ij} \vee (Pv_{i-1,j-1} \wedge \exists \ell \in [1,i-1] : Eq_{\ell j} \wedge (\forall x \in [\ell, i-2] : Pv_{x,j-1})) \end{aligned}$$

  Väite seuraa nyt induktio-oletuksesta.

□

Lemman 2.7.12 kaava näyttää mahdottomalta laskea vakioajassa edes yhdelle bitille, saati sitten bittirinnakkaisesti koko vektorille Xh_{*j}. Se on kuitenkin mahdollista, mutta bittioperaatioita tarvitaan lisää:

- Merkitään xor-operaatiota symbolilla $\veebar$: $0 \veebar 1 = 1 \veebar 0 = 1$ ja $0 \veebar 0 = 1 \veebar 1 = 0$.
- Tulkitaan bittivektorit kokonaislukujen bittiesitykseksi siten, että vähiten merkitsevä bitti on positiossa 1. Bitit listataan eniten merkitsevästä alkaen. Esimerkiksi $B_{1\ldots m} = B_m B_{m-1} \cdots B_1$.
- Käytetään yhteenlaskua bittioperaationa. Esim. $0001 + 0111 = 1000$.

Lemma 2.7.13 *Merkitään $X = Xh_{*j}$, $E = Eq_{*j}$, $P = Pv_{*,j-1}$ ja olkoon $Y = (((E \wedge P) + P) \veebar P) \vee E$. Tällöin $X = Y$.*

Todistus. Kirjoitetaan X_i:n määritelmä ensin toiseen muotoon:

- $X_i = 1$, jos ja vain jos
  a) $E_i = 1$ tai
  b) $\exists \ell \in [1,i] : E_{\ell\ldots i} = 00\cdots01 \wedge P_{\ell\ldots i-1} = 11\cdots1$.
- $X_i = 0$, jos ja vain jos
  c) $E = 00\cdots0$ tai
  d) $\exists \ell \in [1,i] : E_{\ell\ldots i} = 00\cdots01 \wedge P_{\ell\ldots i-1} \neq 11\cdots1$.

Osoitetaan tapauskohtaisesti, että $Y_i = X_i$:

a) Y:n määritelmän loppu "$\vee E$" takaa, että $Y_i = 1$.

b) Olkoon b bitti P_i eli $P_{\ell \ldots i} = \mathtt{b11...1}$ ja c bittien $1 \ldots, \ell - 1$ yhteenlaskusta mahdollisesti tuleva muistibitti.

$$\begin{array}{rcl}
 & & \quad i \qquad\quad \ell \\
E_{\ell \ldots i} & = & \mathtt{00...01} \\
P_{\ell \ldots i} & = & \mathtt{b1...11} \\
(E \wedge P)_{\ell \ldots i} & = & \mathtt{00...01} \\
((E \wedge P) + P)_{\ell \ldots i} & = & \bar{\mathtt{b}}\mathtt{0...0c} \\
(((E \wedge P) + P) \veebar P)_{\ell \ldots i} & = & \mathtt{11...1}\bar{\mathtt{c}} \\
Y = ((((E \wedge P) + P) \veebar P) \vee E)_{\ell \ldots i} & = & \mathtt{11...11}
\end{array}$$

missä $\bar{\mathtt{b}}$ on b:n negaatio.

c) Koska kaikilla bittivektoreilla B, $0 \wedge B = 0$ ja $0 + B = B$, saadaan $Y = (((0 \wedge P) + P) \veebar P) \vee 0 = (P \veebar P) \vee 0 = 0$.

d) Tarkastellaan kohdan b) laskentaa. Sen ideana on, että yhteenlaskussa positiossa ℓ syntyvä muistibitti kantautuu aina positioon i asti ja tuottaa sinne bitin $\bar{\mathtt{b}}$. Erona on nyt, että ainakin yksi bitti P_k, $\ell \leq k < i$, on 0. Jos $k = \ell$, niin $P_\ell = (E \wedge P)_\ell = 0$ ja muistibittiä ei synny (riippumatta bitistä c). Jos taas $k > \ell$, muistibitti pysähtyy positioon k. Kummassakin tapauksessa $((E \wedge P) + P)_i = \mathtt{b}$ ja siten $Y_i = 0$.

□

- Määritellään $Peq[a]$, $a \in \Sigma$, on bittivektori, jossa on 1 positiossa i jos ja vain jos $p_i = a$. Tällöin $Eq_{*j} = Peq[s_j]$.

Algoritmi 2.7.14 *Myers*
Syöte: P, S, k, $|P| \leq w$, missä w on tietokoneen sanan pituus
Tuloste: P:n likimääräiset esiintymäkohdat k virheellä S:ssä

Laske $Peq[a]$ jokaiselle $a \in \Sigma$;
$Pv := 1^m$; $Mv := 0$; $Sc := m$;
for $j := 1$ **to** n **do**
  $Eq := Peq[S_j]$;
  $Xh := (((Eq \wedge Pv) + Pv) \veebar Pv) \vee Eq$;
  $Ph := Mv \vee \neg(Xh \vee Pv)$;
  $Mh := Pv \wedge Xh$;
  $Xv := Eq \vee Mv$;
  $Pv := (Mh << 1) \vee \neg(Xv \vee (Ph << 1))$;
  $Mv := (Ph << 1) \wedge Xv$;
  $Sc := Sc + Ph(m) - Mh(m)$;
  if $Sc \leq k$ **then** *write* j

Lause 2.7.15 *Olkoon hahmo $P = p_1 \cdots p_m$ ja teksti $S = s_1 \cdots s_n$. Likimääräinen hahmonsovitus k virheellä voidaan ratkaista Algoritmilla 2.7.14 ajassa $\mathcal{O}(\lceil m/w \rceil n)$, missä w on tietokoneen sananpituus bitteinä.*

Todistus. On selvää, että Algoritmi 2.7.14 toimii ajassa $\mathcal{O}(n)$ kun $m \leq w$. Algoritmia voidaan helposti simuloida pidemmillä hahmoilla käyttäen useaa tietokonesanaa vektorien Pv, Eq, jne., esittämiseen. □

Huomautus 2.7.16 *Rajoittamalla laskenta kullakin sarakkeella viimeiseen arvoon k asti (kuten Algoritmissa 2.7.6), voidaan Algoritmia 2.7.14 tehostaa toimimaan odotusarvoisesti $\mathcal{O}(\lceil k/w \rceil n)$ ajassa.*

Huomautus 2.7.17 *Monessa laskennan mallissa oletetaan, että $w = \Omega(\log n)$ (jotta muistiosoite mahtuu vakiomäärään sanoja). Tällöin siis Algoritmin 2.7.14 aikavaatimus on oikeastaan $O(mn/\log n)$. Muitakin samanlaisia algoritmeja on. Masekin ja Patersonin "neljän venäläisen" menetelmään perustuvalla algoritmilla [11] aikavaatimus on jopa $O(mn/\log_\sigma^2 n)$.*

2.8 Seulontamenetelmät

- Edellä on kehitetty algoritmeja, joiden pahimman tapauksen aikavaativuudet ovat $O(mn)$, $O(kn)$, $O(\lceil m/w \rceil n)$, tms.
- Monissa sovelluksissa ylläolevat aikavaativuudet ovat vielä liian suuria; tarvetta on lineaarisille tai jopa alilineaarisille ratkaisuille.
- *Seulontamenetelmillä* voidaan saavuttaa nämä tavoitteet odotusarvoisessa tapauksessa, kun sallittu virheiden määrä k on tarpeeksi pieni.
- Seulontamenetelmien yleinen kehikko:
  - *Vaihe 1*: Käytetään algoritmia F, jonka aikavaatimus on $O(f(m,n,k) + |Pocc|)$, *seulomaan* tekstistä joukko $Pocc = \{j\}$ *mahdollisia* esiintymien loppukohtia.
  - *Vaihe 2*: Käytetään algoritmia G, jonka aikavaatimus on $O(g(m,n,k,|Pocc|))$, *tarkistamaan* onko kukin $j \in Pocc$ *todellisen esiintymän* loppukohta.
- Tavoitteena on valita F ja G siten, että kokonaissuoritusaika $O(f(m,n,k) + |Pocc| + g(m,n,k,|Pocc|))$ on mahdollisimman pieni.

- Käytännössä algoritmi G on esim. dynaaminen ohjelmointi tai Myersin tehostettu versio.
- Jäljelle jää valita algoritmi F siten, että se toimii mahdollisimman nopeasti, mutta samalla tulostaa mahdollisimman vähän *vääriä* esiintymäkandidaatteja.

Odotusarvoisesti lineaariaikainen seulontamenetelmä

R. Baeza-Yates ja C. Perleberg. Fast and Practical Approximate String Matching. *Information Processing Letters*, 59:21–27, 1996.

- Olkoon P hahmo, S teksti, ja k yläraja virheiden määrälle.
- Etsitään siis kaikkia S:n positioita j, joihin päättyy jokin S:n osajono $S_{j'\ldots j}$ siten, että editointietäisyys $D(P, S_{j'\ldots j}) \leq k$.
- Johdetaan ongelman ratkaisemiseksi yksinkertainen seulontamenetelmä, joka perustuu seuraavaan lemmaan:

Lemma 2.8.1 *Jos hahmo P pilkotaan $k+1$:een palaseen $P = P^1P^2 \cdots P^{k+1}$, niin ainakin yksi palasista löytyy muuttumattomana minkä tahansa likimääräisen esiintymän sisältä.*

Todistus. Kukin muutos, lisäys tai poisto voi muuttaa enintään yhtä palasista. Siten k editointioperaatiota voi muuttaa enintään k:ta palasta. □

- Käytetään Lemmaa 2.8.1 seuraavasti:
  - Vaihe 1: Etsitään kaikkien $k+1$:n palasen tarkkoja esiintymiä tekstistä Aho-Corasick automaatilla.
  - Vaihe 2: Tarkistetaan dynaamisella ohjelmoinnilla vaiheessa 1 löytyneet mahdolliset esiintymäkohdat.
- Algoritmin odotusarvoisen käyttäytymisen analyysi:
  - Vaihe 1 toimii aina ajassa $O(n + |Pocc|)$ (vakioaakkostolla), joten voidaan keskittyä vaiheen 2 analyysiin.

- Merkitään luvulla $E(r,n)$ r:n pituisen hahmon (palasen) odotusarvoista esiintymien määrää tekstissä, jonka pituus on n; oletetaan, että hahmon ja tekstin merkit on riippumattomasti ja tasaisesti valittu aakkostosta, jonka koko on σ.
- Saadaan arvio
$$E(r,n) = \frac{n-r+1}{\sigma^r} \leq \frac{n}{\sigma^r}.$$
- Olkoon hahmo P jaettu palasiin $P = P^1 P^2 \cdots P^{k+1}$, missä kukin $|P^i| \geq r$ ja r on suurin mahdollinen. Siis $r = \lfloor m/(k+1) \rfloor$.
- Käyttämällä odotusarvon lineaarisuutta, odotusarvo $k+1$:n palasen yhteenlasketulle esiintymämäärälle, on korkeintaan $(k+1)E(r,n)$. Tämä on siis odotusarvo luvulle $|Pocc|$, eli vaiheen 1 antaman mahdollisten esiintymien joukon koolle.
- Jos käytetään dynaamista ohjelmointia tarkistusvaiheessa, yhden esiintymäkandidaatin tarkistamisen aikavaatimus on $O(m^2)$. Siten
$$g(m,n,k,|Pocc|) = \mathcal{O}(m^2(k+1)E(r,n)) = \mathcal{O}(m^3 \frac{n}{\sigma^r}),$$
missä on käytetty arviota $k \leq m$.
- Tarkistusvaihe toimii odotusarvoisesti lineaarisessa ajassa jos $m^3 \frac{n}{\sigma^r} \leq c \cdot n$, jollain vakiolla $c > 0$. Ratkaisemalla tämä r:n suhteen saadaan
$$r \geq 3 \log_\sigma m - \log_\sigma c = \Omega(\log_\sigma m)$$
- Siis tarkistusvaihe voidaan ratkaista odotusarvoisesti lineaarisessa ajassa kun
$$k+1 \leq m/r = \mathcal{O}\left(\frac{m}{\log_\sigma m}\right)$$

Lause 2.8.2 *Olkoon hahmo $P = p_1 \cdots p_m$ ja teksti $S = s_1 \cdots s_n$. Yllä kuvattu seulontamenetelmä toimii odotusarvoisesti ajassa $\mathcal{O}(n)$, kun $k = \mathcal{O}(m/\log_\sigma m)$.*

- Vaihe 2 voidaan itse asiassa ratkaista odotusarvoisesti ajassa $\mathcal{O}(n/m)$, kun yllä $c \cdot n$ korvataan $c \cdot n/m$:llä. Ehto $k = \mathcal{O}(m/\log_\sigma m)$ pysyy ennallaan.

- Vaihe 1 voidaan myös ratkaista odotusarvoisesti ajassa $\mathcal{O}(n\frac{\log_\sigma m}{r}) = \mathcal{O}(nk\frac{\log_\sigma m}{m})$ käyttäen optimaalista monen hahmon etsimiseen tarkoitettua algoritmia (esim. MultiBDM [4]).

- Saadaan siis algoritmi, joka on odotusarvoisesti lineaarinen, kun $k = \Theta(m/\log_\sigma m)$, ja alilineaarinen, kun $k = o(m/\log_\sigma m)$.
- Ongelmaan on kehitetty useita muitakin odotusarvoisesti lineaarisia/alilineaarisia ratkaisuja.
- Chang & Marr [2] todistavat, että $\Omega(n\frac{(k+\log_\sigma m)}{m})$ on alaraja ongelman odotusarvoiselle vaativuudelle. He antavat myös optimaalisen algoritmin tapaukseen $k < \frac{m}{3} - O(\frac{m}{\sqrt{\sigma}})$.

2.9 Likimääräinen hahmonsovitus k epätäsmäyksellä

- k epätäsmäystä -ongelma on yksinkertaistus k virhettä -ongelmasta:
  - Vain muutos-operaatiot sallitaan eli käytetään editointietäisyyden sijasta Hamming-etäisyyttä
    $$D_H(A, B) = |\{i \mid A_i \neq B_i\}| ,$$
    joka on määritelty vain kun $|A| = |B|$.
  - Etsitään siis tekstin S positioita j (esiintymien *alkupositioita*), joille pätee $D_H(S_{j...j+m-1}, P) \leq k$.
- Ongelma voidaan ratkaista käyttämällä samantapaisia tekniikoita kuin Kappaleessa 1.8 käytettiin merkkijononsovitukseen jokerimerkeillä; Abrahamson [1] on näyttänyt miten ongelma voidaan ratkaista ajassa $\mathcal{O}(n\sqrt{m}\log m \log\log m)$.
  - Tuloksella ei ole juuri käytännön merkitystä (triviaali-algoritmi on nopeampi kaikilla kuviteltavissa olevilla syötteillä).
  - Teoreettisena tuloksena tämä kuitenkin näyttää, että ongelma voidaan ratkaista $o(mn)$ ajassa.
- *Triviaali-algoritmi:*

(1) **for** $j := 1$ **to** $n - m + 1$ **do**
(2) $\quad$ $mismatch := 0$; $i := 1$;
(3) $\quad$ **while** $i \leq m$ **and** $mismatch \leq k$ **do**
(4) $\quad\quad$ **if** $s_{j+i-1} \neq p_i$ **then** $mismatch := mismatch + 1$;
(5) $\quad\quad$ $i := i + 1$;
(6) $\quad$ **if** $i = m + 1$ **and** $mismatch \leq k$ **then** *write* j;

  - Triviaali-algoritmi toimii pahimmassa tapauksessa ajassa $O(mn)$.
  - On kuitenkin helppo näyttää, että odotusarvoinen suoritusaika on $\mathcal{O}(kn)$.

- Seuraavassa esitetään, miten ongelma voidaan ratkaista $\mathcal{O}(kn)$ ajassa myös pahimmassa tapauksessa.

Galil-Giancarlo-algoritmi

Z. Galil & R. Giancarlo. Improved String Matching with k Mismatches. *SIGACT News*, 17:52–54, 1986.

Algoritmin idea.

- Toimitaan kuten triviaalialgoritmissa. Verrataan hahmoa ensin jonoon $S_{1...m}$, sitten jonoon $S_{2...m+1}$, jne. Kussakin positiossa etsitään $k+1$ ensimmäistä epätäsmäystä tai esiintymän tapauksessa havaitaan, että niin montaa epätäsmäystä ei löydy.
- Samoin kuin Knuth-Morris-Pratt algoritmissa, vältetään tutkimasta jo tutkittua osaa tekstistä uudelleen.

Algoritmi tarkemmin.

- Tarkastellaan hahmon vertailua osajonoon $S_{j...j+m-1}$.
- Olkoon *end* viimeinen tekstin positio, jota on verrattu hahmoon, ja olkoon *start* se alkupositio, joka aiheutti tämän vertailun.
- Siis tekstin osajonoa $S_{start...end}$ on verrattu hahmon alkuosaan $P_{1...end-start+1}$. Niiden välillä on enintään $k+1$ epätäsmäystä, koska vertailua ei kauempaa jatketa. (Esiintymän tapauksessa epätäsmäyksiä on vähemmän kuin $k+1$.) Olkoon MM lista näistä epätäsmäyskohdista: $MM = \{j' \in [start, end] \mid S_{j'} \neq P_{j'-start+1}\}$.
- Olkoon $MM(i)$ lista epätäsmäyksistä hahmon alkuosan $P_{1...m-i+1}$ ja loppuosan $P_{i...m}$ välillä. Siis $MM(i) = \{i' \in [1, m-i+1] \mid P_{i'} \neq P_{i+i'-1}\}$.
- Epätäsmäykset $S_{j...j+m-1}$:n ja P:n välillä (tai ainakin $k+1$ ensimmäistä niistä) löydetään järjestyksessä kahdessa vaiheessa.

1. Osajonon $S_{j...end}$ ja hahmon alkuosan $P_{1...end-j+1}$ väliset epätäsmäykset löydetään selaamalla listoja MM ja $MM(j - start + 1)$ rinnakkain ja etsimällä positioparit (j', i'), $j \leq j' \leq end$ ja $i' = j' - j + 1$, joilla joko $j' \in MM$ tai $i' \in MM(j - start + 1)$.
   - Jos $j' \notin MM$ ja $i' \notin MM(j - start + 1)$ (eli paria (j', i') ei löydetä), pätee $S_{j'} = P_{j'-start+1} = P_{i'}$ eli täsmäys.
   - Jos $j' \notin MM$ ja $i' \in MM(j - start + 1)$, pätee $S_{j'} = P_{j'-start+1} \neq P_{i'}$. Epätäsmäys.
   - Jos $j' \in MM$ ja $i' \notin MM(j - start + 1)$, pätee $S_{j'} \neq P_{j'-start+1} = P_{i'}$. Epätäsmäys.
   - Jos $j' \in MM$ ja $i' \in MM(j - start + 1)$, pätee $S_{j'} \neq P_{j'-start+1} \neq P_{i'}$. Verrataan merkkejä $S_{j'}$ ja $P_{i'}$.
2. Jos osasta $S_{j...end}$ epätäsmäyksiä löytyi vähemmän kuin $k + 1$ jatketaan vertaamista positiosta $end+1$ eteenpäin triviaalialgoritmilla, kunnes $k + 1$ epätäsmäystä on löydetty tai päädytään hahmon loppuun. Tällöin vaiheen lopussa päivitetään $start$, end ja MM.

Analyysi

- Vaiheessa 2 jokainen askel triviaalialgoritmissa siirtää viimeistä tutkittua kohtaa end eteenpäin tekstissä. Täten triviaali-algoritmin ajamiseen käytetty kokonaisaika on $\mathcal{O}(n)$.
- Vaiheen 1 pareja (j', i'), joilla $j' \in MM$ on enintään $k + 1$ kappaletta, koska $|MM| \leq k + 1$.
- Jokainen löydetty pari (j', i'), jolla $j' \notin MM$ on epätäsmäys, joten niitä on enintään $k + 1$ kappaletta.
- Siis vaiheen 1 aikavaatimus on $\mathcal{O}(k)$.
- Vaiheessa 1 tarvittavat listat $MM(i)$ voidaan laskea ajassa $\mathcal{O}(m^2)$ ja tallettaa tilaan $\mathcal{O}(km)$ (koska vain $2k + 2$ ensimmäistä epätäsmäystä tarvitaan).

Lause 2.9.1 *Galil-Giancarlo algoritmi ratkaisee k epätäsmäystä -ongelman ajassa $O(m^2 + kn)$ ja tilassa $O(km)$.*

- Ongelmaan on kehitetty muitakin $O(kn)$ algoritmeja, mutta ne ovat huomattavasti monimutkaisempia, ja käytännössä hitaampia (ensimmäinen $O(kn)$ algoritmi on [8]).

- Myöhemmin nähdään, että Galil-Giancarlo-algoritmi voidaan implementoida siten, että $O(kn)$ aika ja $O(m)$ tila riittää. Tämä tehdään korvaamalla listat $MM(i)$ tietorakenteella, joka vie tilaa vain $\mathcal{O}(m)$ ja voidaan muodostaa $\mathcal{O}(m)$ ajassa.

- Kokeissa Galil-Giancarlo-algoritmi on noin kaksi kertaa *hitaampi* kuin triviaalialgoritmi (satunnaisella syötteellä).