JBIG2-tiivistysstandardi

Sami Gröhn

22.11.2000

Joensuun yliopisto
Tietojenkäsittelytiede
Kandidaatintutkielma

TIIVISTELMÄ

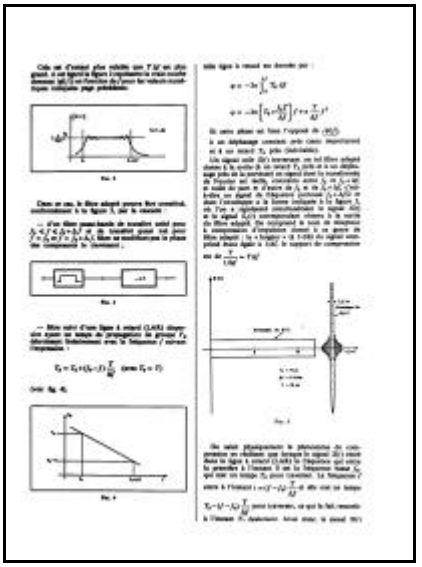
Tiivistyksen tarkoituksena on saada kuvan sisältämä tieto mahdollisimman pieneen tilaan mahdollisimman hyvälaatuisena. Tehokkaalla tiivistyksellä saadaan alkuperäinen kuva tallennettua kymmeniä kertoja pienempään tilaan kuvan laadun kärsimättä. Tässä tutkielmassa tutustutaan tiivistysprosessin kahteen osaan: mallintamiseen ja koodaukseen. Lisäksi esitellään yksi häviötön binäärikuvien tiivistysstandardi, JBIG2. Siitä käydään läpi erilaiset tiivistysmenetelmät kuvien sisältämälle tekstille ja muille osille.

Avainsanat: häviötön kuvantiivistys, tilastollinen tiivistys, kontekstimallintaminen, aritmeettinen koodaus, QM-koodaaja, segmentointi, merkkien täsmäys.

SISÄLTÖ

1 JOHDANTO

Tiivistyksen tarkoituksena on saada kuvan sisältämä tieto mahdollisimman pieneen tilaan mahdollisimman hyvälaatuisena. Tehokkaalla tiivistyksellä saadaan alkuperäinen kuva tallennettua kymmeniä kertoja pienempään tilaan kuvan laadun kärsimättä, mikä on tärkeää digitaalisessa muodossa olevien kuvien määrän jatkuvasti kasvaessa.



Kuva 1.1. Esimerkki tyypillisestä skannatusta binäärikuvasta.

Tässä tutkielmassa tutustutaan JBIG2-standardiin kaksiväristen binäärikuvien tiivistämisessä [JBIG Committee, 2000]. JBIG2 on vuonna 1993 valmistuneen JBIG-standardin [JBIG, 1993] seuraaja ja sisältää JBIG:n lähes sellaisenaan. Suurin muutos on erilainen koodausmenetelmä kuvan tekstialueille. Koska itse standardi antaa paljon liikkumavaraa käytännön toteutuksille, otetaan tässä kantaa myös erilaisiin toteutusvaihtoehtoihin.

Luvussa kaksi perehdytään JBIG2:n kannalta tärkeisiin tiivistyksen peruskäsitteisiin. Tiivistys voidaan jakaa kahteen osaan, mallintamiseen ja koodaukseen. Erilaisten mallintamismenetelmien lisäksi käydään läpi aritmeettinen koodaus [Rissanen, Langdon 1979].

Kolmas luku käsittelee segmentointia, jonka tarkoituksena on erotella kuvasta erityyppistä tietoa sisältävät alueet. Tekstialue sisältää nimensä mukaisesti tekstiä, yksittäisten kirjainten

muodostamia joukkoja. Rasterikuva alue sisältää yleensä valokuvia, mutta sen koodausta ei käsitellä tarkemmin. Muut kuvan osat luokitellaan grafiikaksi.

Neljännessä luvussa käydään läpi JBIG-standardissa esitelty menetelmä binäärikuvien tiivistämiseksi, jota käytetään JBIG2:ssa grafiikan koodaukseen. Menetelmä perustuu kontekstimalliin sekä aritmeettisen koodaajan binääriselle datalle tehtyyn muunnelmaan, QM-koodaajaan [Pennebaker, Mitchell 1988].

Viides luku käsittelee tekstin koodaamista. Tekstin koodaus perustuu tekstin symbolien bittikartat sisältävään sanakirjaan sekä tekstialueeseen, johon talletetaan sanakirjan indeksit ja symbolien sijainnit. Myös symbolien täsmäys eli sanakirjassa olevan symbolin vertaaminen koodattavaan symboliin esitellään.

Kuudennessa luvussa esitellään JBIG2-standardin kaksi erilaista tiedostorakennetta, peräkäisrakenne sekä suorasaanti [JBIG Committee]. Lisäksi kuvatiedoston sisältämien tiedostosegmenttien sisältö selvitetään lyhyesti.

2 TILASTOLLINEN TIIVISTYS

Tiedon tiivistäminen perustuu kaavaan [Fränti 1999]:

$$Tieto = tietosisältö + redundanssi$$

Tiivistyksen tarkoituksena on poistaa *redundanssi (redundancy)*, jolla tarkoitetaan tietosisällön ylimääräistä toistoa. Symbolien informaatioarvoa mitataan sen *entropialla (entropy)*:

$$H(x) = -\log_2 p(x) \tag{1}$$

missä x on symboli ja $p(x)$ sen esiintymistodennäköisyys. Entropia $H(x)$ kertoo bittimäärän, joka tarvitaan symbolin x koodaamiseen, jotta päästäisiin optimaaliseen tulokseen [Shannon 1948]. Todennäköisyysjakauman kokonaisentropia saadaan laskemalla yksittäisten symbolien entropioiden odotusarvo H, missä m on aakkoston symbolien lukumäärä:

$$H = -\sum_{x=1}^{m} p(x) \log_2 p(x) \tag{2}$$

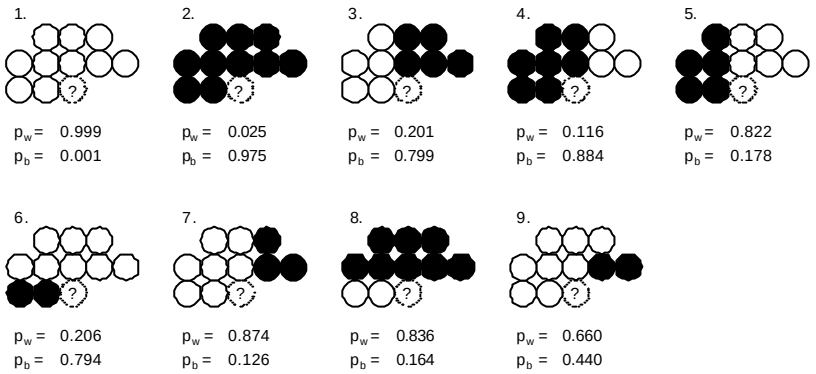
Entropia määrää tiivistyksen alarajan (bittiä/symboli) johon mallia noudattava tiedosto on mahdollista tiivistää. Koodaus on *optimaalinen (optimal)*, jos koodin pituus on sama kuin entropia. Kuvassa 2.1 on esimerkki yksinkertaisesta todennäköisyysmallista sekä sen entropiasta.

symboli	ASCII	H(x)	p(x)
a	00	1	0.5
b	01	2	0.25
c	10	3	0.125
d	11	3	0.125

Kuva 2.1. Esimerkki todennäköisyysmallista neljälle symbolille. Mallin kokonaisentropiaksi saadaan 1.75.

Kontekstimallissa (context model) hyödynnetään pikseleiden keskinäisiä riippuvuuksia. Esimerkiksi jos jonkin kuvan pikselin kaikki naapuripikselit ovat mustia, myös koodattava pik-

seli on musta hyvin suurella todennäköisyydellä. Todennäköisyysjakauma riippuu siis suuresti siitä, missä *kontekstissa (context)* se esiintyy. Siksi jokaiselle kontekstille käytetään erilaista mallia. Kuva, josta malli muodostetaan, käydään läpi rivi kerrallaan vasemmalta oikealle ja jokaisen pikselin pikselin naapuristosta kerätään tietoa. Kontekstien lukumääräksi tulee kaikki mahdolliset naapuripikseleiden kombinaatiot, jolloin esimerkiksi käytettäessä kymmenen pikselin naapuristoa, saadaan 2^{10} = 1024 erilaista kontekstia. Kuvassa 2.2 on esitelty yhdeksän yleisintä 10-pikselin kontekstia kuvalle 1.1.



Kuva 2.2. Yhdeksän yleisintä 10-pikselin kontekstia kuvalle 1.1. [Ageenko 2000].

Mallin muodostamiseksi on useita vaihtoehtoja. Yksinkertaisin on *staattinen malli (static model)*, jossa samaa oletettua mallia käytetään kaikille kuville. Staattisen mallin ongelmana on todellisen syötteen ja mallin eroavaisuudet, jotka heikentävät tiivistystehoa.

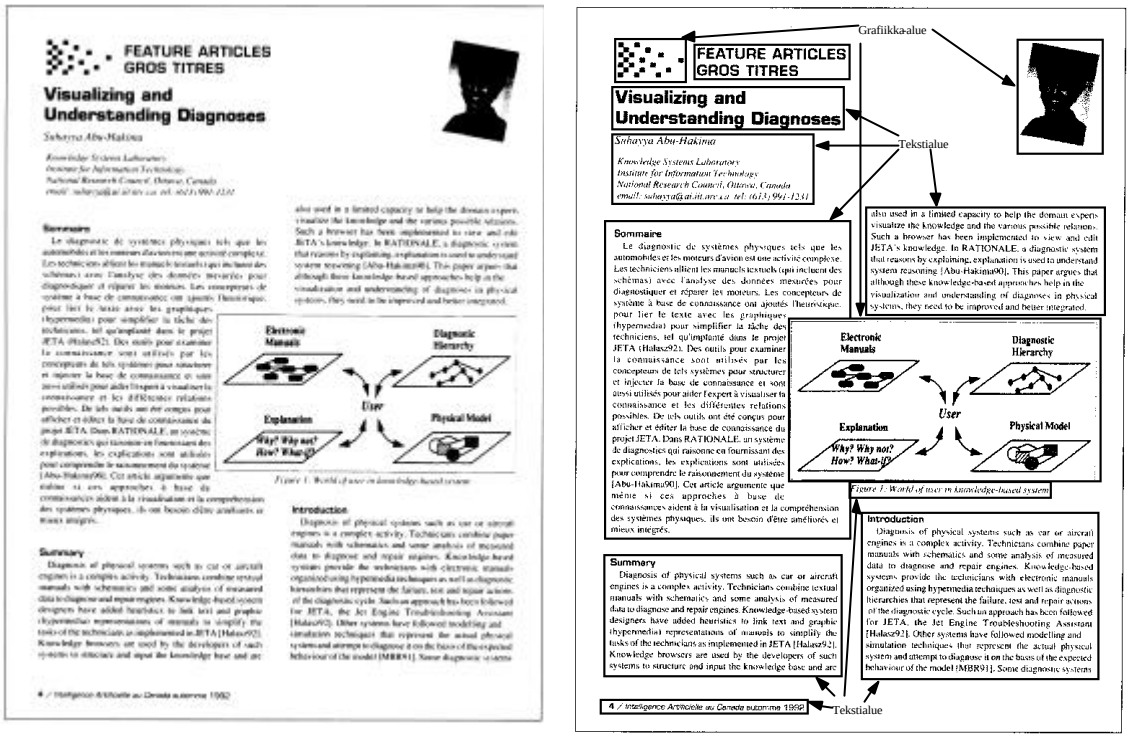
Astetta kehittyneemmässä *semi-adaptiivisessa mallintamisessa (semi-adaptive model)* koodattava kuva käydään läpi kahdesti: ensimmäisessä vaiheessa kuvasta muodostetaan malli, joka talletetaan koodattavan kuvan lisäksi. Toisessa vaiheessa kuva koodataan muodostetun mallin perusteella.

Adaptiivisessa mallintamisessa (adaptive model) kuvasta ei kerätä etukäteen mitään tietoa, vaan koodaaja päivittää mallia koodauksen edetessä. Koodauksen alkuvaiheessa käytetty malli voi olla tehoton, mutta se *sopeutuu (adapt)* kuvaan kun jo koodattuja pikseleitä päästään hyödyntämään. Koodattava kuva tarvitsee käydä läpi vain kerran, eikä itse mallia tarvitse tallettaa, koska myös dekoodaaja päivittää mallia koodauksen edetessä.

Koodaus voidaan suorittaa optimaalisesti *aritmeettisella koodaajalla (arithmetic coder)*, jota sekä JBIG että JBIG2 käyttävät [Rissanen, Langdon 1979]. Aritmeettisen koodauksen perusidea on esittää koko koodattava tiedosto yhtenä pienenä reaalilukuvälinä välillä [0,1]. Koodaus aloitetaan jakamalla väli [0,1] kahteen aliväliin mallin antamien mustan ja valkoisen pikselin todennäköisyyksien mukaan. Seuraavaksi valitaan koodattavaa pikseliä vastaava aliväli, joka jaetaan edelleen kahteen osaan mallin perusteella. Prosessia toistetaan jokaiselle koodattavalle pikselille, jolloin väli pienenee kunnes lopullinen väli määrittää koodattavan kuvan yksikäsitteisesti.

3 KUVAN SEGMENTOINTI

Jotta kuvan erityyppiset alueet voitaisiin koodata juuri niille suunnitelluilla menetelmillä, on ne voitava jotenkin erotella toisistaan. Tällöin puhutaan kuvan *segmentoinnista (segmentation)*. Segmentoinnin tarkoituksena on jakaa kuva suorakulmaisiin *alueisiin (region)*, joista kukin sisältää pelkästään joko tekstiä tai grafiikkaa (kuva 3.1). Itse standardi ei kuitenkaan määrittele miten segmentointi on suoritettava vaan ainoastaan sen miten segmentoitu kuva talletetaan tiedostoon, jotta dekoodaaja osaisi sen purkaa.

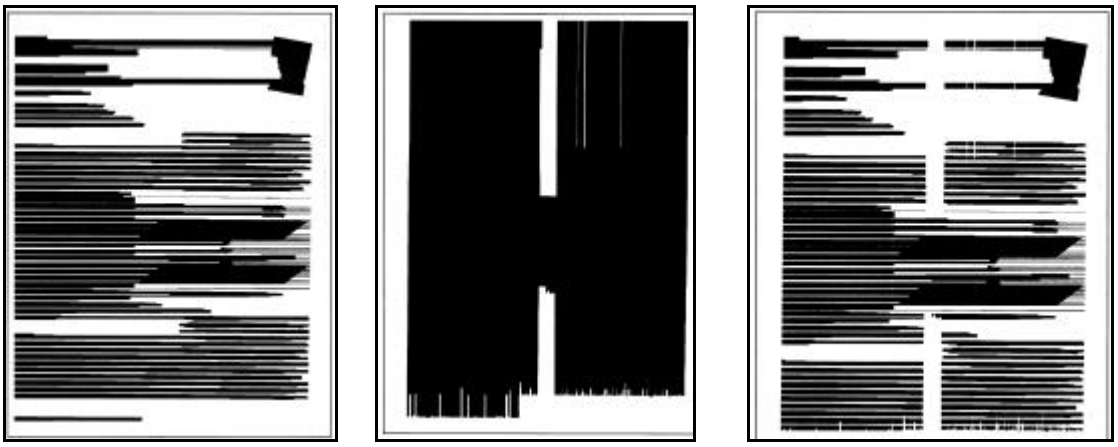


Kuva 3.1. Vasemmalla alkuperäinen kuva joka sisältää erityyppistä tietoa. Oikealla esimerkki saman kuvan segmentoinnista alueisiin.

3.1 Segmentointimenetelmät

Segmentoinnin toteuttamiseen on olemassa kaksi vastakkaista lähestymistapaa: *ylhäältä alas (top-down)* ja *alhaalta ylös (bottom-up)*. Ylhäältä alas-menetelmät tarkastelevat kuvaa kokonaisuudessaan ja etsivät siitä yhteenkuuluvia kokonaisuuksia. Alhaalta ylös-menetelmissä merkit yhdistetään toistuvasti yhä suuremmiksi kokonaisuuksiksi.

Alhaalta ylös-menetelmät aloittavat yleensä pehmentämällä *(smearing) kuvan*. Pehmennyksessä väritetään mustaksi kaikki kahden mustan pikselin väliset pikselit, jos mustien pikseleiden keskinäinen etäisyys on alle ennalta määrätyn raja-arvon. Pehmennys voidaan tehdä erikseen vaaka- ja pystysuunnissa, jonka jälkeen kaksi saatua kuvaa yhdistetään loogisella AND-operaatiolla. Witten, Moffat ja Bellin [1999] mukaan menetelmä toimii kohtuullisesti kuville, joissa on yksinkertainen sijoittelu, kuten kuva 1.1. Esimerkki pehmennysoperaation toiminnasta on kuvassa 3.2.

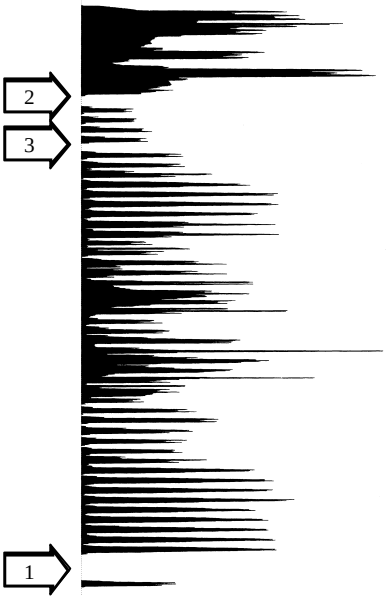


a **b** **c**

Kuva 3.2. Pehmennysoperaatio kuvalle 3.1. Pehmennys vaakasuunnassa (a), pystysuunnassa (b) sekä edelliset yhdistettynä (c). [Witten, Moffat, Bell 1999].

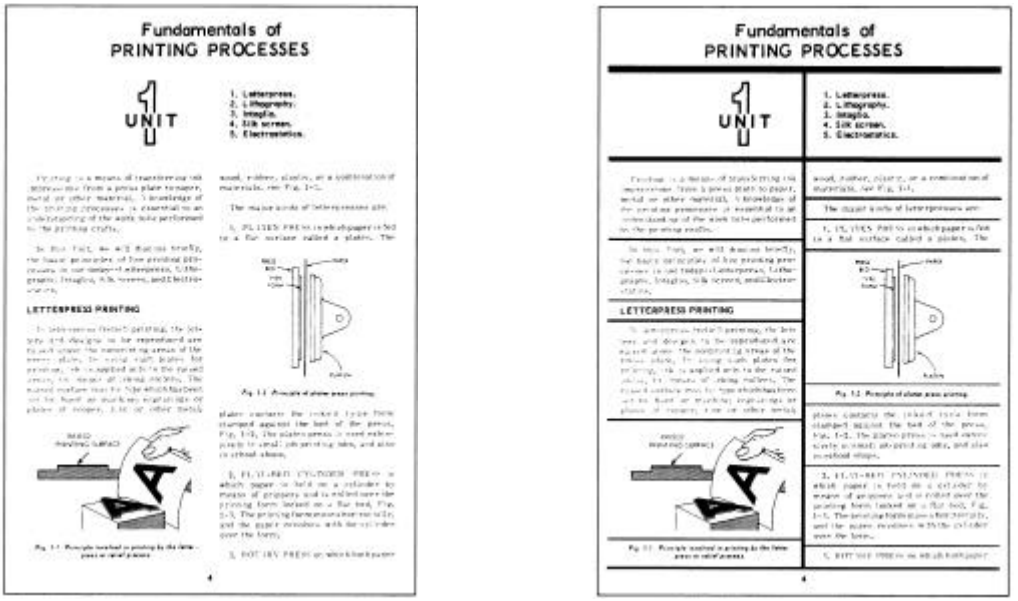
Toinen vaihtoehto suorittaa pehmennys on tutkia jokaisen pikselin naapuristoa ja värittää pikseli mustaksi jos naapuristossa on vähintään ennalta määrätty määrä mustia pikseleitä. Tämän jälkeen kuva käydään uudelleen läpi ja väritetään pikseli mustaksi, jos naapuristossa on yksikin musta pikseli. Tämä menetelmä on hitaampi mutta se antaa yleensä paremman lopputuloksen [Witten, Moffat, Bell 1999].

Ylhäältä alas-menetelmä jakaa kuvan lohkoihin rekursiivinen X-Y leikkaus algoritmilla [Witten, Moffat, Bell 1999]. Jokaisessa vaiheessa lasketaan kuvan jokaisella rivillä ja jokaisessa sarakkeessa olevien mustien pikseleiden lukumäärä. Tuloksena saadaan kaksi kuvan 3.3 tyylistä pylväsdiagrammia, joista toisessa on rivisummat ja toisessa sarakesummat.



Kuva 3.3. Kuvan 3.1 mustien pikseleiden lukumäärä riveittäin sekä kolme ensimmäistä leikkauskohtaa.

Diagrammeista etsitään levein valkoisten pikseleiden muodostama väli, josta kuva leikataan kahteen osaan. Sen jälkeen kuva leikataan seuraavaksi leveimmästä välistä ja tätä toistetaan kunnes vähintään ennalta määrätyn levyisiä välejä ei enää ole jäljellä (kuva 3.4).



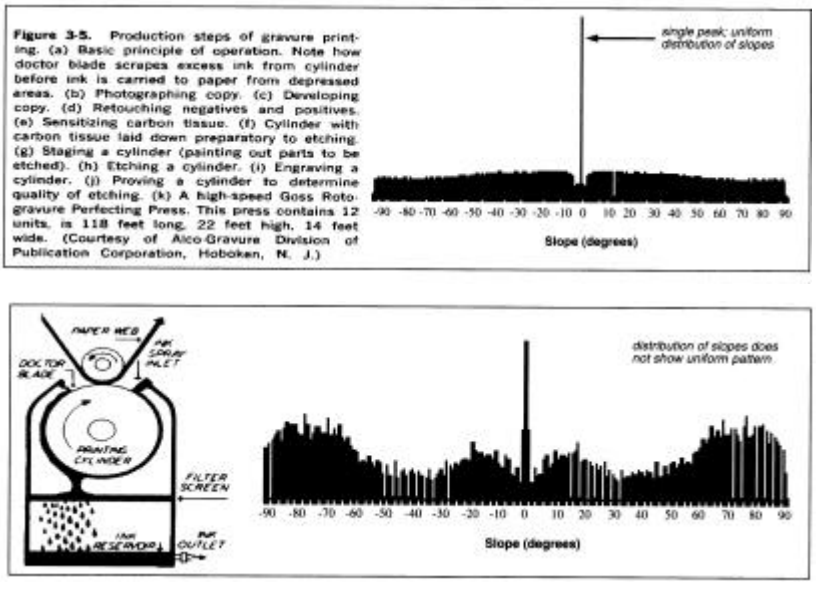
Kuva 3.4. Rekursiivinen X-Y leikkaus-algoritmi. Vasemmalla alkuperäinen kuva ja oikealla leikkausten paikat. Paksummat viivat tarkoittavat aikaisemmassa vaiheessa suoritettuja leikkauksia [Witten, Moffat, Bell 1999].

Algoritmi toimii hyvin yksinkertaisen rakenteen sisältäville kuville, kuten kuva 3.4, mutta esimerkiksi kuvan 3.1 se segmentoi huonosti, koska siinä alueiden reunat eivät ole samalla tasalla pysty –ja vaakasuunnassa. Ylhäältä alas- ja alhaalta ylös-menetelmät voidaan yhdistää suorittamalla kuvan pehmennys ennen rekursiivista X-Y leikkausta, jolloin päästää yleensä parempaan lopputulokseen kuin suorittamalla menetelmät erikseen.

3.2 Segmenttien luokittelu

Kun kuva on segmentoitu alueisiin, on alueet vielä luokiteltava niiden sisältämän tiedon mukaan tekstialueisiin ja grafiikkaan. Luokittelu on tärkeä vaihe, koska epäonnistunut luokittelu johtaa selvästi heikompaan tiivistystulokseen. Jos segmentoinnin tuloksena on saatu eroteltua yksittäiset tekstikappaleet, kuten kuvassa 3.3, voidaan luokitteluperusteena käyttää alueiden kokoa ja muotoa. Tekstialueet ovat yleensä leveitä ja matalia, kun taas grafiikka-alueet ovat epäsäännöllisempiä.

Luokittelu voidaan tehdä myös *merkkien (mark)* perusteella. Yhdellä merkillä tarkoitetaan alueella olevaa yhtenäistä mustien pikseleiden yhdistämää aluetta. Merkkien lukumäärä alueella on yleensä suuri tekstille ja pieni grafiikalle. Witten, Moffat ja Bell [1999] esittävät menetelmän, jossa jokaiselle kahden merkin kombinaatiolle lasketaan niiden välinen kulma. Näin saadusta jakaumasta muodostetaan *kallistumadiagrammi (slope diagram)*, kuten kuvassa 3.5. Tekstille jakauma on tekstin säännöllisestä sijoittelusta johtuen hyvin tasainen ja grafiikalle epäsäännöllinen.



Kuva 3.5. Esimerkki kallistumadiagrammista tekstille (ylempi) ja grafiikalle (alempi).

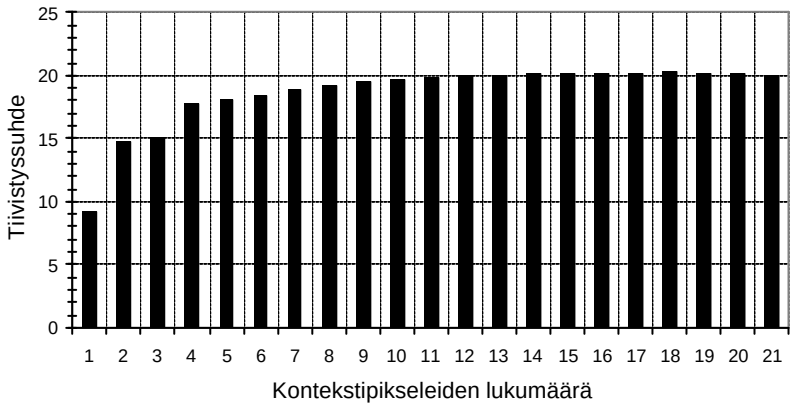
4 GRAFIIKAN KOODAUS

JBIG2:n sisältämä grafiikan koodaus on lähes sama kuin JBIG1, jossa kuva tiivistetään pikseli kerrallaan. Tiivistettävän alueen rivit käydään läpi ylhäältä alas ja jokaisen rivin pikselit vasemmalta oikealle. Koodattavan pikselin naapuripikseleiden yhdistelmä määrittelee kontekstin ja jokaiselle kontekstille mustien ja valkoisten pikseleiden todennäköisyysjakauma päätetään adaptiivisesti jo koodattujen pikseleiden perusteella.

```
for i:=1 to N do
  for j:=1 to M do
      c:=MuodostaKonteksti(i,j);
      KoodaaPikseli(x_ij,c);
  endfor
endfor
```

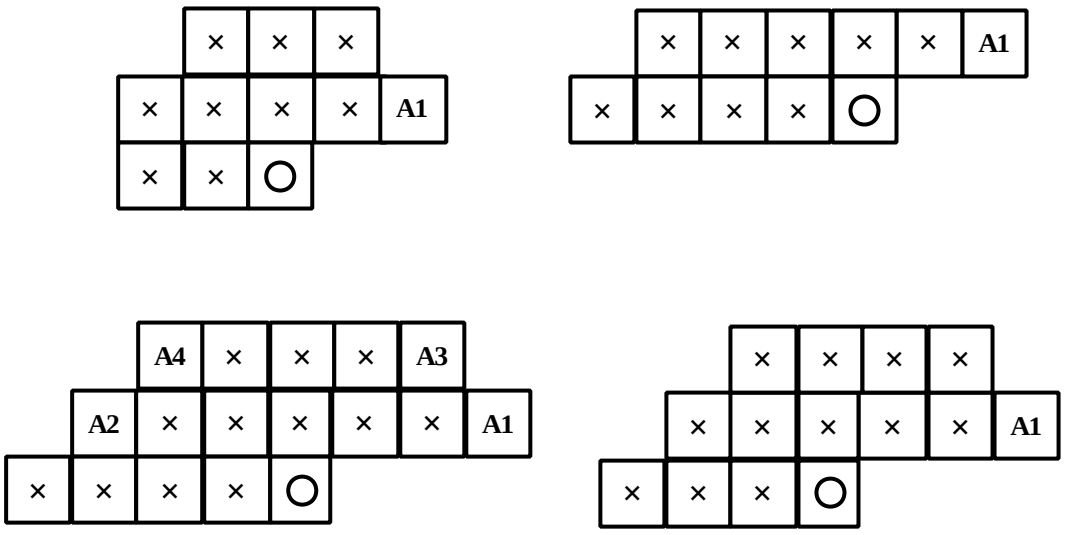
Kuva 4.1. JBIG-algoritmin toiminta [JBIG, 1993].

Mitä enemmän naapuristossa on pikseleitä, sitä tarkempi todennäköisyysmalli on mahdollista saavuttaa. Suureen malliin mukautuminen kestää kuitenkin pidempään, koska erilaisia kontekseja on enemmän, minkä vuoksi malli ei voi olla mielivaltaisen suuri. Yleensä suuremmille kuville voidaan käyttää suurempia naapuristoja. Kuvassa 4.2 on vertailtu eri kokoisten kontekstien tiivistystehoa viidelle 5000×5000-kokoiselle testikuvalle.



Kuva 4.2. Kontekstipikseleiden lukumäärän vaikutus tiivistyssuhteeseen [Ageenko 2000].

Binäärikuville kontekstimalli toimii erityisen hyvin, koska suuristakin pikselinaapuristoista muodostuu suhteellisen vähän erilaisia konteksteja. Kuvassa 4.3 on esitelty JBIG2:ssa käytettävät pikselinaapuristot.



Kuva 4.3. JBIG2:ssa käytettävät pikselinaapuristot. Ylemmät kaksi naapuristoa sisältyvät myös JBIG:een. Pikselit A1-A4 tarkoittavat adaptiivisia pikseleitä.

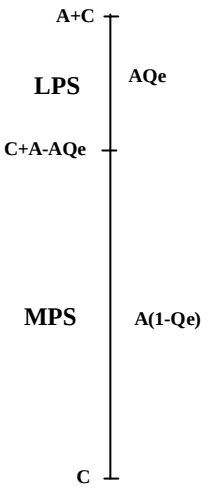
Adaptiivinen pikseli (A1-A4) tarkoittaa pikseliä, jonka paikka naapuristossa voi muuttua kesken koodauksen. Tällöin dekoodaajalle lähetetään tieto muutoksesta. Pikseleiden siirtämisen tarkoituksena on parantaa tiivistystehoa rasterikuvissa, joissa on säännöllinen rasterin rakenne. Esimerkiksi sanomalehtien harmaasävyiset valokuvat muodostuvat lähemmin tarkasteltaessa pienistä säännöllisesti sijaitsevista mustista pisteistä.

4.1 QM-koodaaja

QM-koodaaja on aritmeettisen koodaajan binääriselle tiedolle suunniteltu muunnos, jota käytetään JBIG:ssä. [Pennebaker, Mitchell 1988]. Se ei ole täysin optimaalinen, mutta sopeutuu aritmeettista koodaajaa nopeammin kuvaan ja toimii nopeammin koska kaikki kertolaskut on eliminoitu.

QM-koodaajassa välin koodaamisessa käytetään kolmea muuttujaa, alarajaa *C*, välin pituutta *A* ja vähemmän todennäköisen symbolin todennäköisyyttä *Qe*. Jos koodattavan pikselin väri

on sillä hetkellä todennäköisempi kahdesta väristä, kutsutaan sitä nimellä *MPS (more probable symbol)* ja päinvastaisessa tapauksessa sitä sanotaan *LPS*:ksi *(less probable symbol)* (kuva 4.4).



Kuva 4.4. Välin jakaminen QM-koodaajassa.

Välin pituuden laskiessa alle 0.75:n se kaksinkertaistetaan bitinsiirto-operaatioiden avulla. Näin ollen välin pituus A on aina välillä 0.75 - 1.5 ja koodattaessa pikseliä QM-koodaaja olettaa sen kertolaskujen eliminoimiseksi aina vakioksi 1.0. Tämän seurauksena väli muuttuu seuraavasti:

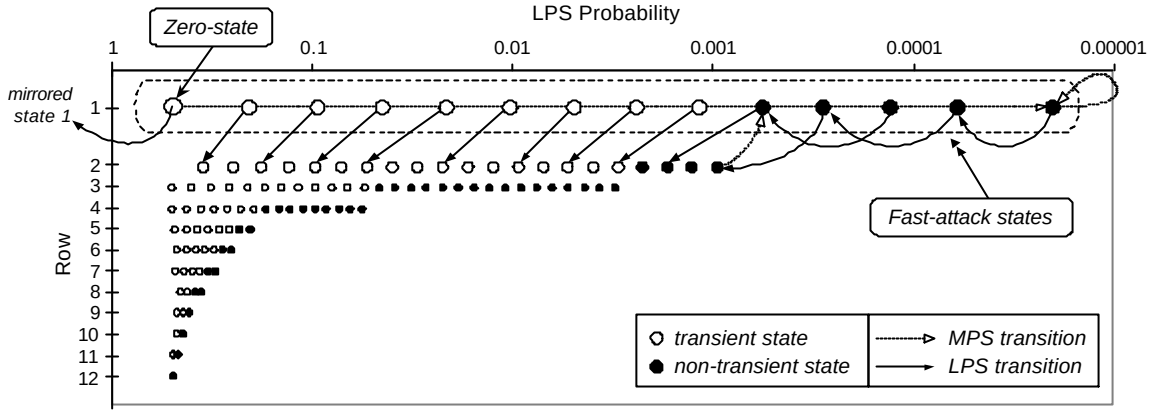
Koodattaessa MPS:

$$C := C \qquad (3)$$
$$A := A \cdot (1 - Qe) = A - A \cdot Qe \approx A - Qe$$

Koodattaessa LPS:

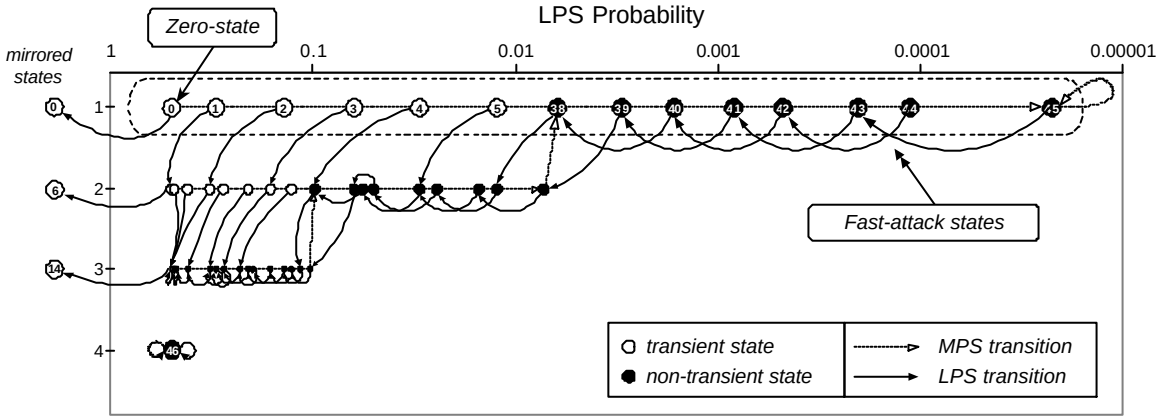
$$C := C + A \cdot (1 - Qe) = C + A - A \cdot Qe \approx C + A - Qe \qquad (4)$$
$$A := A \cdot Qe \approx Qe$$

QM-koodaaja sisältää myös mallinnuksen, joka ei perustu aritmeettisen koodaajan tapaan jo koodattuihin pikseleihin vaan tila-automaattiin (kuva 4.5), jonka avulla kuvaan sopeutuminen nopeutuu. Koodaus alkaa *nollatilasta (zero-state)*. Jokaisesta tilasta on siirtymät kahteen muuhun tilaan koodattavan pikselin värin perusteella. Koodauksen edetessä automaatti vakiintuu *stabiileihin (non-transient)* tiloihin.



Kuva 4.5. QM-koodaajan käyttämä tila-automaatti todennäköisyyksien ennustamiseen [Ageenko 2000].

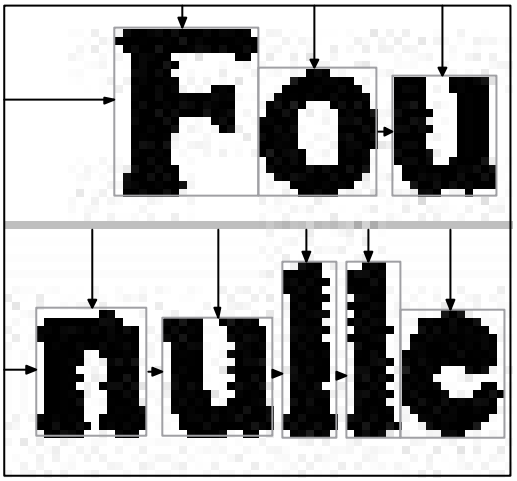
QM-koodaajasta edelleen kehitettyä MQ-koodaajaa käytetään JBIG2:ssa grafiikan koodaukseen. Tila-automaatin tilojen lukumäärää on vähennetty 226:sta 94:een (kuva 4.6), mikä nopeuttaa kuvaan sopeutumista entisestään.



Kuva 4.6. MQ-koodaajan tila-automaatti.

5 TEKSTIALUEIDEN KOODAUS

JBIG2:n näkyvin uudistus JBIG:een on erillinen koodausmenetelmä tekstialueille. Tekstialueet koodataan *sanakirjan (dictionary)* sekä siihen viittaavan *tekstialueen (text region)* avulla. Tyypillisesti tekstissä samat kirjaimet esiintyvät toistuvasti. Sen sijaan että koodaisimme jokaisen saman kirjaimen esiintymän bittikartan, valitsemme yhden edustavan *esiintymän (instance)*, symbolin, ja laitamme sen sanakirjaan. Tekstialueessa on talletettu koodattavan kirjaimen sijainti suhteessa edelliseen kirjaimeen (kuva 5.1) sekä kirjainta vastaava sanakirjan indeksi. Koodattava alue jaetaan pystysuunnassa vakiokorkuisiin *raitoihin (stripe)*, ja jokaisen raidan kirjaimet koodataan erikseen. Tällä tavalla kirjainten y-koordinaatit saadaan pienemmiksi, jolloin ne pystytään tallettamaan pienemmällä bittimäärällä. Sijainnin lisäksi talletetaan täsmäävän sanakirjasymbolin indeksi.

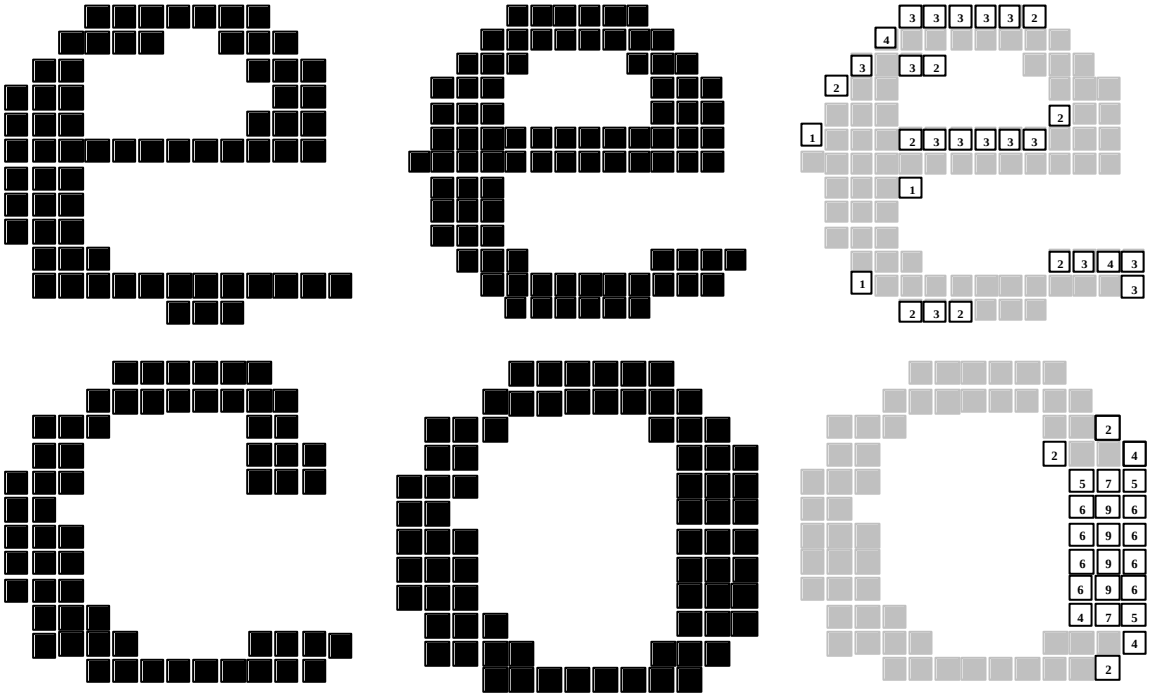


Kuva 5.1. Kirjainten sijainnin koodaus. X-koordinaatti koodataan suhteessa edelliseen kirjaimeen ja y-koordinaatti siirtymänä raidan yläreunasta.

5.1 Merkkien täsmäys

Tekstin koodauksessa oleellisessa osassa on *merkkien täsmäys (pattern matching)*, jonka avulla päätetään ovatko sanakirjassa oleva ja koodattava kirjain samat. Witten, Moffat ja Bell [1999] esittelevät kaksi täsmäysmenetelmää: *globaali täsmäys (global matching)* ja *paikallinen täsmäys (local matching)*.

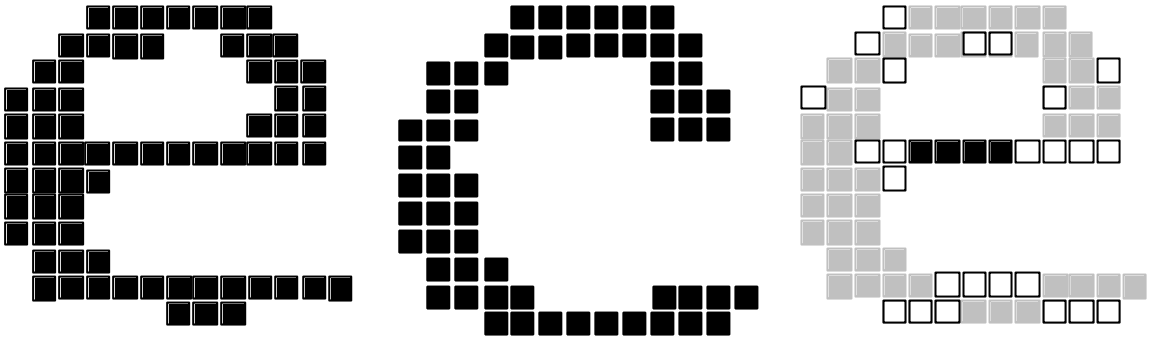
Globaalissa täsmäyksessä tutkitaan kahden kirjaimen eroavien pikseleiden kokonaismäärää, jonka perusteella muodostetaan *virhekartta (error map)*. Virheitä painotetaan sen mukaan missä yhteydessä ne esiintyvät. Esimerkiksi kuvassa 5.2 ylärivillä verrataan kahta e-kirjainta esittävää bittikarttaa. Kuvien välillä eroavien pikseleiden lukumäärä (29) on suurempi kuin alarivillä verrattaessa c- ja o-kirjaimia toisiinsa (23). Alemmassa virhekartassa eroavat pikselit esiintyvät kuitenkin ryhmässä, jolloin niillä on suurempi painoarvo. Jokaiselle eroavalle pikselille virhekartassa lasketaan painotettu arvo sen 3×3 naapuristossa olevien eroavien pikseleiden summana. Näin laskettuna ylärivin e-kirjaimille saadaan virheeksi yhteensä 75 ja alarivin c-ja o-kirjainten virheeksi 131. Täsmäys hyväksytään jos virhe on pienempi kuin ennalta sovittu raja-arvo. Tässä tapauksessa raja-arvon tulisi olla välillä 75-131 jotta ylempi täsmäys hyväksyttäisiin ja alempi hylättäisiin.



Kuva 5.2. Kirjainten e ja e sekä c ja o globaali täsmäys käyttäen virhekarttaa [Witten, Moffat, Bell 1999].

Paikallisessa täsmäyksessä virhekarttaan merkitään eroavat pikselit (valkoiset pikselit kuvan 5.3 virhekartassa) sekä erityisesti sellaiset pikselit, jotka ovat jommassakummassa täsmättävistä bittikartoissa mustia ja toisessa kokonaan valkoisten pikseleiden ympäröimiä (mustat pikselit kuvan 5.3 virhekartassa). Täsmäys hylätään, jos mustalla pikselillä virhekartassa on

naapureinaan enemmän mustia pikseleitä kuin ennalta määrätty raja-arvo tai samalla kohdalla sijaitseva pikseli on toisessa bittikartassa kokonaan valkoisten tai mustien pikseleiden ympäröimä. Kuva 5.3 havainnollistaa kuinka e- ja c-kirjainten paikallinen täsmäys johtaa täsmäyksen hylkäämiseen.



Kuva 5.3. Paikallinen merkkien täsmäys e- ja c-kirjaimille.

5.2 Täsmättyjen merkkien koodaus

JBIG2:ssa on kaksi tapaa koodata tekstialueita: *PM&S (pattern matching and substitution)* sekä Howardin 1997 esittämä *SPM (soft pattern matching)*.

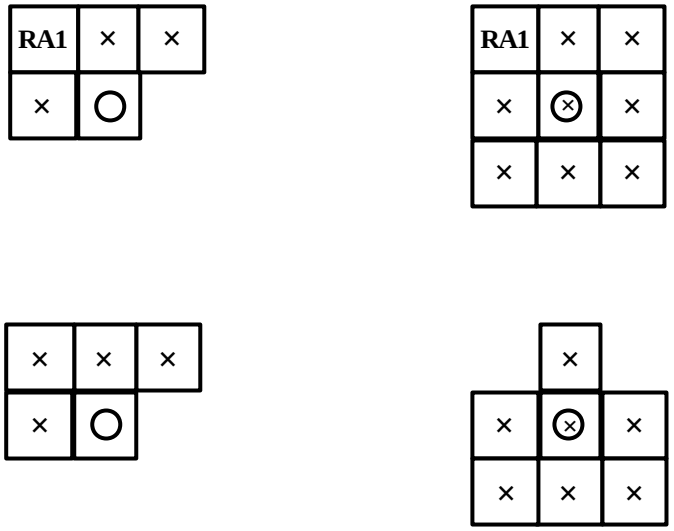
PM&S-menetelmässä (kuva 5.4) jokaista koodattavaa kirjainta verrataan sanakirjassa oleviin symboleihin. Jos koodattava kirjain muistuttaa tarpeeksi sanakirjassa olevaa symbolia, koodataan täsmäävän symbolin indeksi sanakirjassa. Muussa tapauksessa koodattava merkki tulkitaan uudeksi symboliksi ja sen bittikartta koodataan JBIG:llä. Koodauksen jälkeen symboli lisätään sanakirjaan. PM&S-menetelmä mahdollistaa suuren häviöllisen tiivistystehon jos kirjainten täsmäyksessä ollaan tarpeeksi ahneita. Tästä kuitenkin seuraa väistämättä virheitä, jos koodattava kirjain täsmää väärään symboliin.

PM&S-algoritmi	SPM-algoritmi

segmentoi alue symboleihin
for jokaiselle symbolille **do**
  etsi symbolille täsmäys sanakirjasta
  if täsmäys löytyy
  koodaa täsmäävän symbolin indeksi
  else
  koodaa bittikartta
  lisää uusi symboli sanakirjaan
  endif
  koodaa symbolin sijainti
endfor
	segmentoi alue symboleihin
	for jokaiselle symbolille **do**
	  etsi symbolille täsmäys sanakirjasta
	  if täsmäys löytyy
	  koodaa täsmäävän symbolin indeksi
	  koodaa bittikartta täsmäävän symbolin avulla
	  mahdollisesti lisää uusi symboli sanakirjaan
	  else
	  koodaa bittikartta
	  lisää uusi symboli sanakirjaan
	  endif
	  koodaa symbolin sijainti
	endfor

Kuva 5.4. JBIG2:n merkkienkoodausalgoritmit.

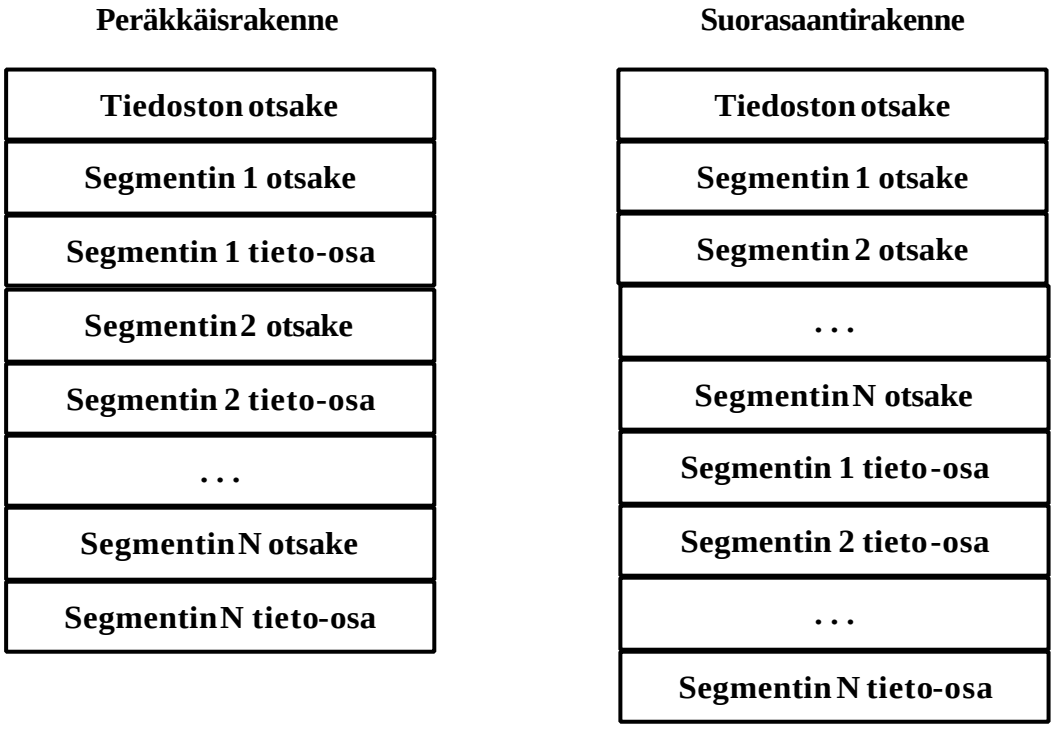
Täsmäysvirheiden välttämiseksi voidaan käyttää häviötöntä SPM-menetelmää. Siinä koodataan kirjaimen sijainnin ja sanakirjaindeksin lisäksi *tarkennustietoa (refinement data)*, jonka avulla alkuperäinen merkki voidaan palauttaa. Käytännössä tämä tarkoittaa alkuperäisen bittikartan koodaamista indeksin lisäksi. Tarkennustieto koostuu koodattavan kirjaimen pikseleistä, jotka koodataan käyttäen konstekstina koodattavan merkin jo koodattuja pikseleitä sekä täsmäävän merkin pikseleitä (kuva 5.5). Tällä tavoin sanakirjan symbolien informaatiota voidaan tehokkaasti hyödyntää ja bittikartta saadaan yleensä tiivistettyä huomattavasti pienempään tilaan kuin jos täsmäystä ei olisi suoritettu ja olisi käytetty JBIG:iä. SPM ei tarvitse tarkkaa symbolien täsmäysmenetelmää kuten PM&S, koska täsmäysvirhe johtaa vain huonompaan tiivistyssuhteeseen.



Kuva 5.5. SPM:ssä käytettävät vaihtoehtoiset pikselinaapuristot. Vasemmalla koodattavan symbolin pikselit, oikealla täsmäävän symbolin pikselit.

6 JBIG2:N TIEDOSTORAKENNE

JBIG2:lla tiivistetty tiedosto koostuu useista erillisistä osista, *segmenteistä (segment)*, joista kukin sisältää tietoa alkuperäisestä kuvasta. Segmenttejä on olemassa useita eri tyyppejä vastaamaan kuvan eri osia, sillä esimerkiksi grafiikka ja teksti koodataan eri tavoilla. Standardi määrittelee kaksi vaihtoehtoista tiedostorakennetta, *peräkkäisrakenteen (sequential)* sekä *suorasaannin (random-access)*, joissa segmentin osat on järjestetty eri tavoilla (kuva 6.1).



Kuva 6.1. JBIG2:n vaihtoehtoiset tiedostorakenteet.

Suorasaantirakenteessa kaikki segmenttien otsakkeet on talletettu tiedoston alkuun. Tämä mahdollistaa tieto-osien purkamisen halutussa järjestyksessä, sillä otsakkeiden lukemisen jälkeen jokaisen tieto-osan sijainti tiedostossa voidaan helposti laskea. Peräkkäisrakenteessa segmentit puretaan aina samassa järjestyksessä kuin ne on talletettu.

JBIG2:n tiedostorakenteessa jokaista koodattua kuvaa vastaa yksi kuvan kanssa saman kokoinen sivu. Koska samassa tiedostossa voi olla useita sivuja, voidaan usean kuvan sarjat tallettaa samaan tiedostoon.

6.1 Segmentin rakenne

Jokainen segmentti koostuu kahdesta osasta, *otsakkeesta (header)* sekä itse *tieto-osasta (data)*. Lisäksi tieto-osan alussa on vielä erikseen *tieto-otsake (data header)*, jonka muoto riippuu segmentin tyypistä.

Segmentin otsake on kaikilla segmenttityypeillä saman muotoinen. Se sisältää seuraavat tiedot:

Kuva 6.2. Tiedostosegmentin rakenne.

Segmenttityyppejä ovat tekstialueiden symbolien tallettamiseen käytettävä *symbolisanakirjasegmentti (symbol dictionary segment)* sekä aluesegmentit *tekstialuesegmentti (text region segment)* ja grafiikan talletukseen käytetty *yleinen alue-segmentti (generic region segment).* Lisäksi jokaista tiedoston sivua kohti on *sivun tiedot-segmentti (page information segment).* Tieto segmentin tieto-osan pituudesta mahdollistaa suorasaannin toteuttamisen otsakkeiden lukemisen jälkeen.

Tieto-otsakkeen muoto riippuu segmenttityypistä. Alue-segmenteillä tieto-otsake sisältää alueen koon ja sijainnin. Yleisestä alueesta talletetaan lisäksi koodaamisessa käytetyn pikselinaapuriston numero (1-4) ja tekstialueesta sen sisältämien kirjainten esiintymien lukumäärä. Symbolisanakirjasta talletetaan koodaamisessa käytetty pikselinaapuristo sekä sanakirjan sisältämien symbolien lukumäärä. Sivusta riittää tallettaa sen koko, sillä sivunumero muodostetaan automaattisesti sen mukaan monesko sivun tiedot-segmentti on kyseessä.

6.2 Tekstialueiden tallennus

Tekstialue jaetaan tiedostossa kahteen eri segmenttiin: symbolisanakirjasegmentti sisältää kirjainten bittikartat tiivistettyinä yksitellen MQ-koodaajalla ja tekstialuesegmentti sisältää yksittäisten kirjainten esiintymät.

Tekstialuesegmentissä jokainen esiintymä sisältää kirjaimen sijainnin tekstialueella, täsmäävän sanakirjaindeksin sekä mahdollisesti SPM-menetelmää käytettäessä syntyneen tarkennustiedon kirjaimesta. Koska tekstialuesegmentti käyttää hyväksi symbolisanakirjan bittikarttoja, teksti-aluesegmentin on *viitattava (reference)* symbolisanakirjasegmenttiin. Kaikki viittaukset ovat sallittuja vain pienempinumeroisiin segmentteihin, joten sanakirjasegmentin on sijaittava tiedostossa ennen sitä käyttäviä tekstialueita. Standardi sallii usean tekstialueen käyttää samaa sanakirjaa, jolloin kirjainten bittikartat riittää koodata vain kerran.

7 YHTEENVETO

Binäärikuvien tiivistämiseen voidaan käyttää tehokkaasti kontekstimallinnukseen perustuvia menetelmiä, koska erilaisten kontekstien lukumäärä on hyvin rajallinen. Tiivistystä voidaan yhä tehostaa jakamalla kuva grafiikkaan ja tekstiin, jotka kumpikin tiivistetään erityisesti niille suunnatuilla menetelmillä.

Tässä tutkielmassa luotiin katsaus tilastolliseen tiivistykseen, adaptiiviseen mallintamiseen sekä kuvan segmentointiin ja segmenttien luokitteluun. Lisäksi esiteltiin eräs näitä menetelmiä käyttävä tiivistysstandardi, JBIG2.

VIITELUETTELO

Ageenko E.I. *Context-Based Compression of Binary Images*. Ph.D. Thesis, Joensuun yliopisto, Joensuu, 2000.

Fränti P.: *Image Compression*, Lecture notes, Joensuun yliopisto, Joensuu, 1999.

Howard P.G.: Text Image Compression Using Soft Pattern Matching. *The Computer Journal*, 40 (2/3):146--156, 1997.

JBIG. ISO/IEC International Standard 11544 *ISO/IEC/JTC1/SC29/WG9*, 1993.

JBIG Committee: JBIG2, Working Draft, Internet WWW-sivu, http://www.jpeg.org/public/jbigpt2.htm (30.10.1999).

Pennebaker W.B., Mitchell J.L.: Probability estimation for the Q-coder. *IBM Journal of Research, Development* 32(6): 737--759, 1988.

Rissanen J., Langdon G.G.: Arithmetic Coding. *IBM Journal of Research and Development* 23(2), 149--162, 1979.

Shannon, C.E.: A Mathematical Theory of Communication. *Bell Systems Technical Journal*, 27, 379--423, 623--656, 1948.

Witten I.H., Moffat A., Bell T.C.: *Managing Gigabytes: Compressing and Indexing Documents and Images (2nd edition)*. Morgan Kaufmann, USA, 1999.