1. SYNTAKTISET JA RAKENTEELLISET MENETELMÄT

Tähän mennessä esitetyissä tunnistusmenetelmissä oletettiin, että hahmot voidaan esittää ja luokitella piirrevektoreiden avulla huomioimatta erityisemmin hahmon rakennetta.

Joissain tapauksissa tunnistus ei onnistu pelkkien piirrevektoreiden avulla, vaan piirteiden väliset suhteet on myös huomioitava.

Piirteiden väliset suhteet voidaan esittää esim. formaalin kielen tai graafin avulla.

Syntaktisia ja rakenteellisia menetelmiä voidaan käyttää luokittelun lisäksi hahmojen analysointiin ja generointiin.

Syntaktiset menetelmät perustuvat yleensä jäsentimiin ja rakenteelliset erilaisiin symbolijonojen ja graafien vertailumenetelmiin.

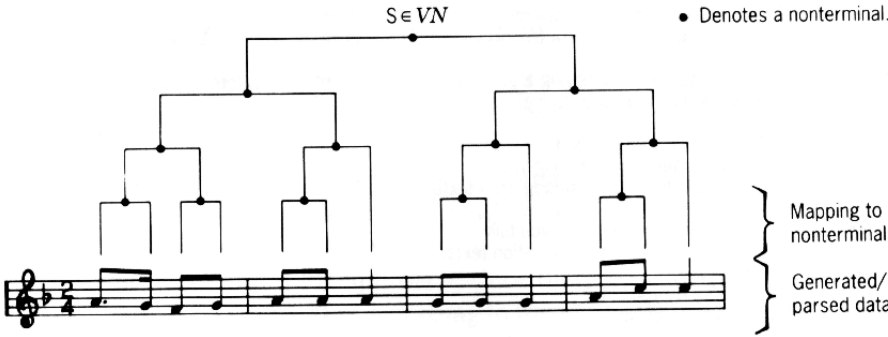
Syntaktisissa ja rakenteellisissa menetelmissä oletetaan usein, että hahmot muodostuvat hierarkisesti yksinkertaisista osista, alihahmoista (esim. kieli - kappaleet - lauseet - sanaluokat - sanat - tavut - kirjaimet - vedot).

Syntaktisessa ja rakenteellisessa hahmontunnistuksessa hahmot jaetaan alihahmoiksi ja näiden alihahmojen väliset suhteet selvitetään.

Eri luokissa voi olla samoja alihahmoja, mutta säännöt alihahmojen välisille suhteille ovat erilaiset.

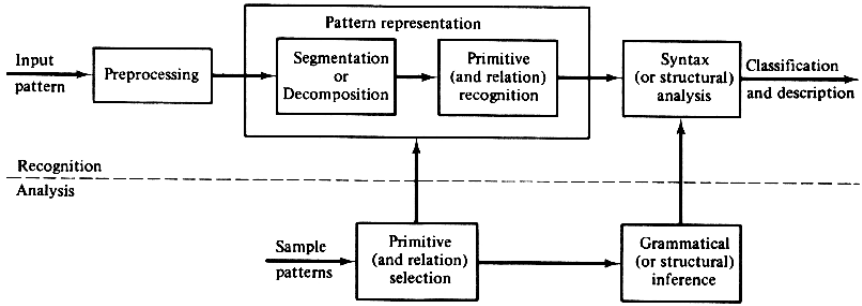
Nämä luokkakohtaiset säännöt saattavat olla ainoa tieto, jonka perusteella hahmot voidaan luokitella.

Esimerkki hahmosta, jolla on sisäinen rakenne:



(Eri melodioissa voi olla saman korkuisia ja kestoisia säveliä (nuotteja), mutta sävelien järjestys on erilainen.)

Kaaviokuva syntaktisesta tunnistusjärjestelmästä:



1.1 Formaalit kielet

Voidaan ajatella, että hahmot ovat formaalin kielen sanoja.

Formaalin kielen kielioppi kertoo millä tavoin hahmojen alihahmoja (symboleja, muuttujia) voidaan yhdistellä suuremmiksi kokonaisuuksiksi (sanoiksi).

Määritellään seuraavaksi **lauserakennekielioppi** ('phrase-structure grammar'):

- Lauserakennekielioppi G koostuu neljästä joukosta $G = (V_N, V_T, P, S)$
- V_N ja V_T ($V_N \cap V_T = \emptyset$) ovat kielen nonterminaali- ja terminaalimuuttujien joukot ja niiden unioni $V_N \cup V_T = V$ on kielen sanasto.
- P on joukko muodostussääntöjä, jotka ovat muotoa $\alpha \rightarrow \beta$, missä α ja β ovat V:n osajoukkoja ja $\alpha \cap V_N \neq \emptyset$
- $S \in V_N$ on aloitusmuuttuja

Joitain hyödyllisiä **merkintöjä**:

- Σ^* on kaikkien äärellisten pituisten symbolijonojen joukko, jotka on muodostettu joukkoon Σ kuuluvista symboleista. λ on tyhjä symbolijono. $\Sigma^+ = \Sigma^* - \{\lambda\}$
- x^n tarkoittaa, että kirjoitetaan symbolijono x n kertaa peräkkäin
- $|x|$ on symbolijonon x pituus
- $J_1 \Rightarrow_G J_2$ tarkoittaa sitä, että symbolijonosta J_1 voidaan *suoraan generoida* symbolijono J_2 eli $J_1 = \omega_1\alpha\omega_2$, $J_2 = \omega_1\beta\omega_2$ ja on olemassa muodostussääntö $\alpha \rightarrow \beta$
- $J_1 \Rightarrow_G^* J_2$ tarkoittaa sitä, että symbolijonosta J_1 voidaan *välillisesti generoida* symbolijono J_2 eli on olemassa symbolijonojen sarja $C_1, \ldots, C_n$ siten, että $J_1 = C_1, J_2 = C_n$ ja $C_i \Rightarrow_G C_{i+1},\ i = 1, \ldots . n-1$. Sarjaa $C_1, \ldots, C_n$ voidaan kutsua J_2:n johtamiseksi ('derivation') J_1:sta

Jos G on lauserakennekielioppi, silloin

$$L(G) = \{x | x \in V_T^* \text{ siten, että } S \Rightarrow_G^* x\} \quad (1)$$

on sitä vastaava *lauserakennekieli*

Mikäli kieleen $L(G)$ kuuluva symbolijono voidaan generoida useammalla kuin yhdellä tavalla, kielioppi G on *moniselitteinen* ('ambiguous'). Hahmontunnistussovellutuksissa pyritään yleensä muodostamaan kielioppi, joka on yksiselitteinen.

Lauserakennekielioppien tyypit

Lauserakennekieliopit voidaan jakaa neljään tyyppiryhmään muodostussääntöjen perusteella:

- Tyyppi 0: **rajoittamattomat kieliopit**

  Tällaiset kieliopit ovat liian yleisiä ollakseen hyödyllisiä ja on erittäin vaikea päätellä onko annettu symbolijono kieliopin mukainen

  (Rajoittamatonta kielioppia vastaa rajoittamaton kieli)

- Tyyppi 1: **kontekstiriippuvat kieliopit**

  Kaikki kieliopin muodostussäännöt ovat muotoa $C_1AC_2 \rightarrow C_1\beta C_2$, missä $A \in V_N$, $C_1, C_2, \beta \in V^*$ ja $\beta \neq \lambda$

  (Kontekstiriippuvaa kielioppia vastaa kontekstiriippuva kieli)

- Tyyppi 2: **kontekstiriippumattomat kieliopit**

  Kaikki kieliopin muodostussäännöt ovat muotoa $A \rightarrow \beta$, missä $A \in V_N$ ja $\beta \in V^+$

Kontekstiriippumattomat kieliopit ovat kaikkien vähiten rajoitettu kielioppityyppi, jolle on olemassa tehokkaita jäsennysmenetelmiä

Muodostussäännöt voidaan esittää myös jäsennyspuun (parsing tree') avulla, joka muodostetaan seuraavasti:

- Puun solmut vastaavat V:hen kuuluvia muuttujia
- Puun juuri vastaa aloitusmuuttujaa S
- Mikäli solmulla on jälkeläisiä, se vastaa nonterminaalimuuttujien joukkoon V_N kuuluvaa muuttujaa
- Mikäli solmut $n_1, \ldots, n_k$ (vastaavat muuttujia $A_1, \ldots, A_k$) ovat solmun n lapsia (vastaa muuttujaa A), silloin on olemassa muodostussääntö $A \to A_1 A_2 \ldots A_k$

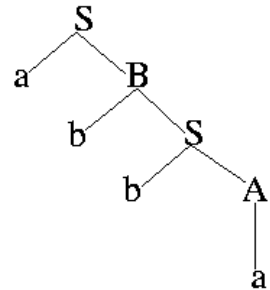
Esimerkki: Olkoot $G = (V_N, V_T, P, S)$, missä $V_N = \{S, A, B\}$,

$V_T = \{a, b\}$ ja

$$\begin{aligned} P = \{ & S \to aB, && S \to bA, \\ & A \to aS, && A \to bAA, \\ & A \to a, && B \to bS, \\ & B \to aBB, && B \to b\} \end{aligned}$$

Nähdään helposti, että kielioppi G on kontekstiriippumaton. Kielioppia vastaava kieli $L(G)$ koostuu symbolijoinoista, joissa on yhtä monta a:ta ja b:tä ja vähintään yksi kumpaakin

Edellä määriteltyyn kieleen kuuluvaa sanaa $abba$ vastaava puu:



$(S \Rightarrow aB \Rightarrow abS \Rightarrow abbA \Rightarrow abba)$

(Kontekstiriippumatonta kielioppia vastaa kontekstiriippumaton kieli)

- Tyyppi 3: **Säännölliset kieliopit** ('finite-state', 'regular')

  Kaikki kieliopin muodostussäännöt ovat muotoa $A \rightarrow aB$ tai $A \rightarrow b$, missä $A, B \in V_N$ ja $a, b \in V_T$. A, B, a ja b ovat kaikki yksittäisiä symboleja

Säännöllisessä kieliopin muodostussäännön molemmilla puolilla voi siis olla korkeintaan yksi nonterminaalisymboli

Säännöllisen kieliopin hyviä puolia: 1) jäsennys voidaan tehdään äärellisellä tilakoneella, 2) kieliopille voidaan muodostaa yksinkertainen graafinen esitys ja 3) on olemassa menetelmiä, joiden avulla voidaan testata kahden kieliopin, G_1 ja G_2 , *ekvivalenttisuus* (ts $L(G_1) = L(G_2)$)

Esimerkki: Tarkastellaan kielioppia $G = (V_N, V_T, P, S)$, missä $V_N = \{S, A\}$, $V_T = \{a, b\}$ ja $P = \{S \rightarrow aA,\ A \rightarrow aA,\ A \rightarrow b\}$. Nähdään helposti, että kielioppi G on säännöllinen ja sitä vastaava kieli on $L(G) = \{a^n b | n = 1, 2, \dots\}$

(Säännöllistä kielioppia vastaa säännöllinen kieli)

- Jokainen säännöllinen kielioppi on kontekstiriippumaton; jokainen kontekstiriippumaton kielioppi on kontekstiriippuva; jokainen kontekstiriippuva kielioppi on rajoittamaton kielioppi

1.2 Formaalin kieliopin oppiminen

Yleensä oletetaan, että formaali kielioppi on 'annettu' eli esim. asiantuntijan muodostama

Kielioppi voidaan myös oppia ('grammatical inference') opetusnäytteistä

Opetusjoukko H voi koostua positiivisista S^+ ja negatiivisista S^- opetusnäytteistä eli $H = \{S^+, S^-\}$

Tavoitteena on oppia kielioppi G_{learn} siten, että näytteet S^+ kuuluvat kieleen $L(G_{\mathsf{learn}})$, mutta näytteet S^- eivät

Voidaan myös 'ektrapoloida' opetusnäytteitä. Jos

$$S^+ = \{ab, aabb, aaabbb, aaaabbbb\}, \tag{2}$$

voidaan päätellä, että

$$L(G_{\mathsf{learn}}) = \{a^n b^n | n \geq 1\} \text{ ja} \tag{3}$$

$$P = \{S \rightarrow ab, S \rightarrow aSb\} \tag{4}$$

Ongelmallista on se, että sama kieli voi vastata useampaa kuin yhtä kielioppia (mikä kielioppi opitaan?) Tämän takia joudutaan asettamaan opittavalle kieliopille lisärajoituksia (esim. valitaan jokin edellä esitellyistä kielioppityypeistä ja mahdollisimman yksinkertaiset muodostussäännöt)

Opetusjoukon kokoon kannattaa kiinnittää huomiota. Usein $|L(G)| = \infty$ ja käytännössä aina $|S^+| \lll |L(G)|$. Kannattaakin valita S^+ siten, että siinä on käytetty kaikkia G:n muodostussääntöjä

Formaali kielioppi voidaan muodostaa opetusjoukon avulla seuraavasti:

- Annettu: S^+ ja S^-
- Oletettu: $G^{(0)}_{\mathsf{learn}} = \{V_N^{(0)}, V_T^{(0)}, P^{(0)}, S^{(0)}\}$ ja P:n, V_N:n ja V_T:n jäsenien muokkaus- ja luomissäännöt
- Tavoite: löytää $G^{(N)}_{\mathsf{learn}}$ siten, että S^+ kuuluu ja S^- ei kuulu kieleen $L(G^{(N)}_{\mathsf{learn}})$

- Oppimisprosessi:
  - aseta $k = 0$
  - valitse opetusnäyte $x^{(i)} \in S^+$. Jos sen jäsennys onnistuu, jatka; muuten muokkaa $G^{(k)}_{\mathsf{learn}}$:ta
  - valitse opetusnäyte $x^{(i)} \in S^-$. Jos sen jäsennys ei onnistu, jatka; muuten muokkaa $G^{(k)}_{\mathsf{learn}}$:ta
  - Jos kaikki opetusnäytteen on käsitelty, lopeta; muuten $k = k + 1$

Em. menetelmän heikkous on se, että P:n, V_N:n ja V_T:n muokkaus-sääntöjen suunnittelu on hankalaa ja jos säännöt eivät ole yksiselitteiset, mahdollisten kielioppien G^{k+1}_{learn} lkm kasvaa nopeasti.

Sääntöjen automaattisessa muokkaamisessa voidaan hyödyntää erilaisia yleistysperiaatteita, esimerkiksi ehto-osan pelkistämistä tai nosto käsitehierarkiassa

1.3 Symbolijonojen tunnistus

Edellä tarkasteltiin, kuinka symbolijonoina esitetyt hahmot voidaan mallintaa (generoida) formaalin kieliopin tai vastaavan tilakoneen avulla.

Seuraavaksi tarkastellaan kuinka voidaan päätellä, onko symbolijono jonkin tietyn kieliopin mukainen tai kuuluuko se johonkin kieleen (tunnistus).

Symbolijonon tunnistus voi perustua joko vertailuun tai jäsentämiseen.

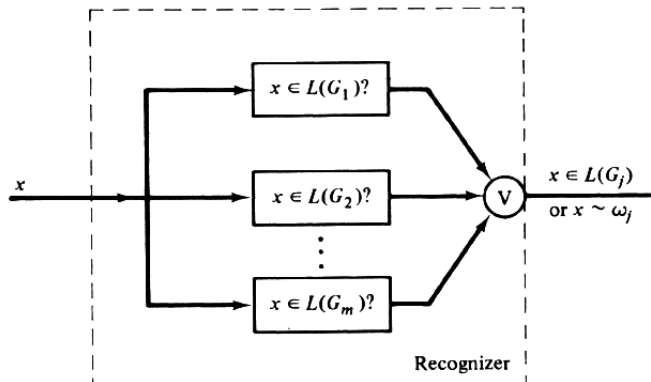
Symbolijonojen jäsennys

Symbolijonon jäsentämisellä tarkoitetaan sitä, että tarkistetaan, onko se syntaktisesti hyväksyttävä eli voidaanko se muodostaa kieliopin sääntöjen mukaisesti.

Jäsentämisen etuna symbolijonojen vertailuun nähden on se, että samalla saadaan selville symbolijonon (hahmon) sisäinen rakenne.

Edellä esitettiin menetelmiä, joiden avulla säännölliset ja kontekstiriippumattomat kieliopit voitiin esittää tilakoneiden ja puiden avulla. Näitä voidaan käyttää symbolijonojen jäsentämiseen.

Kaaviokuva syntaktisesta tunnistuksesta:



Syntaktinen tunnistusongelma voidaan siis formuloida seuraavasti:

$$x \in L(G_i) \text{ jollekin } i = 1, \ldots, M? \tag{5}$$

(x on symbolijono (havainto) ja G_i on luokkaa ω_i vastaava kielioppi)

Tarkastellaan tästä eteenpäin kontekstiriippumattomia kieliä.

Ol., että G on kontekstiriippumaton kielioppi ja x on jokin symbolijono.

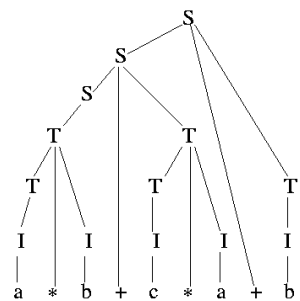
Yritetään muodostaa x:ää vastaavaa jäsennyspuu käyttäen G:n muodostussääntöjä.

Mikäli yritys onnistuu, x kuuluu kieleen $L(G)$; muuten ei

Esimerkki: Tarkastellaan kielioppia $G = (V_N, V_T, P, S)$, missä $V_N = \{S, T\}$, $V_T = \{I, +, *\}$, $I \in \{a, b, c\}$ ja

$$P = \{S \to T, \quad T \to T * I, \quad S \to S + T, \quad T \to I\}$$

Kieleen $L(G)$ kuuluva symbolijono $x = a * b + c * a + b$ voidaan jäsentää seuraavasti:



Ei ole merkitystä muodostetaanko puu lähtien ylhäältä aloitussymbolista (juuresta) vai alhaalta terminaalisymboleista (lehdet). Edellistä tapaa kutsutaan 'top-down'- ja jälkimmäistä 'bottom-up'-jäsentämiseksi

Molemmissa tavoissa pyritään yleensä käsittelemään symbolit järjestyksessä vasemmalta oikealle

Jäsennyksessä käytetään yksi kerrallaan muodostussääntöjä, jotka ovat lokaalisti sopivia

Voi käydä niin, että myöhemmin huomataan jonkin muodostussäännön valinnan johtaneen jäsennysyrityksen epäonnistumiseen, vaikka symbolijono kuuluisikin tarkasteltavaan kieleen. Silloin pitää palata taaksepäin ja tehdä erilainen valinta ('backtracking')

Jotta jäsennys olisi tehokasta, voidaan muodostussääntöjen valintaan käyttää *a priori*-sääntöjä, esim. lokaali sopivuus, johtaako lopulta terminaalisymboleihin, joita esiintyy tarkasteltavassa symbolijonossa ja joita on vähemmän kuin vielä käsittelemättömiä symboleita

Jäsennyksen viemä aika ja tehokkuus riippuvat siitä, kuinka hyvin voidaan välttää väärien muodostussääntöjen valinta

Seuraavaksi tarkastellaan lähemmin 'top-down'- ja 'bottom-up'-lähestymistapoja

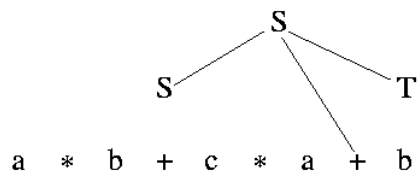
'Top-down'-jäsennys:

- Jäsennys alkaa aloitussymbolista S
- Jäsennys etenee tavoiteorientoituneesti eli tutkitaan voidaanko S hajottaa osatavoitteiksi eli $S \rightarrow X_1 X_2 \ldots X_n$
- Jos X_1 on terminaalisymboli, tarkasteltavan symbolijonon x pitää alkaa tällä
- Jos X_1 on nonterminaalisymboli, otetaan se tarkasteltavaksi osatavoitteeksi ja tutkitaan vastaako x:n alku sitä
- Jos osatavoite X_1 saavutetaan, otetaan X_2 uudeksi tarkasteltavaksi osatavoiteeksi jne

- Mikäli jokin osatavoitteista X_i jää saavuttamatta, kokeillaan uutta aloitusta $S \rightarrow X'_1 X'_2 \ldots X'_n$
- Osatavoitteetkin voidaan hajottaa edelleen uusiksi osatavoitteiksi
- Mikäli jokin osatavoitteista jää saavuttamatta, palataan ylemmälle tasolle ja yritetään uutta osatavoitteiden jakoa
- Huom! vasemmalta oikealle eteneminen ja muotoa $A \rightarrow A\alpha$ olevat rekursiiviset muodostussäännöt voivat yhdessä aiheuttaa ikuisia silmukoita (vältellään näiden sääntöjen käyttöä viimeiseen asti!)
- Tehokas *a priori*-sääntö osatavoitteiden valinnalle: tutkitaan, että vasemmanpuoleisin, käsittelemätön terminaalisymboli voidaan todella johtaa vasemmanpuoleisimmasta, käsittelemättömästä osatavoitteesta (valmiiksi laskettu taulukko!)

- Joitain syitä valita 'top-down'-lähestymistapa:
  - Kielioppi vastaa suoraan hahmojen analysointialgoritmia
  - Takaisin palaamisiin hukattu aika on vähäinen verrattuna terminaalimuuttujien tunnistuksiin käytettyyn aikaan
  - Tavoiteorientoituneisuus on hyödyllistä myös terminaalimuuttujien tunnistukselle

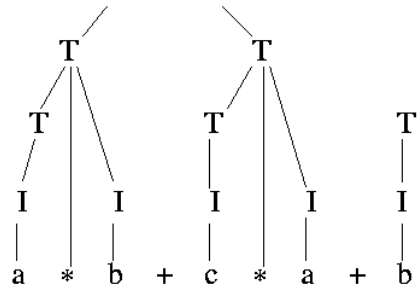
Esimerkki:



'Bottom-up'-jäsennys:

- Jäsennys lähtee liikkeelle tarkasteltavasta symbolijonosta x
- Muodostussääntöjä käytetään 'takaperin'
- Ei osatavoitteita; pyritään vain päätymään lopulta aloitussymboliin
- 'bottom-up'-jäsennys ei ole erityisen tehokas: vääriä valintoja voi olla hyvinkin paljon
- Tehokas *a priori*-sääntö muodostussääntöjen valinnalle: tutkitaan mitkä nonterminaalisymbolit voivat alkaa tietyillä, toisilla nonterminaalisymboleilla (valmiiksi laskettu taulukko!)
- Muotoa $A \rightarrow A\alpha$ olevat rekursiiviset muodostussäännöt eivät ole ongelma
- Mikäli joudutaan umpikujaan, palataan alemmalle tasolle ja kokeillaan vaihtoehtoisia muodostussääntöjä

Esimerkki:



Jäsennys voidaan toteuttaa myös rinnakkaislaskentana: jokaisessa valintatilanteessa monistetaan tarkasteltava ratkaisuyritys, ratkaisuyrityksiä jatketaan rinnakkain, umpikujaan päätyvät yritykset hylätään

'Bottom-up'- ja 'top-down'-lähestymistapojen keskinäinen vertailu on vaikeaa. Toisille kieliopeille 'bottom-up' on tehokkaampi, toisille taas 'top-down'. Lisäksi kieliopin muuttaminen uuteen muotoon vaikuttaa lähestymistapojen tehokkuuksiin

Symbolijonojen vertailu

Yksinkertaisin tapa tunnistaa kieleen kuuluvia symbolijonoja on muodostaa sanakirja: luetellaan (kaikki) kieleen kuuluvat symbolijonot ja verrataan havaittua symbolijonoa näihin

Tunnistus voi perustua esim. k:n lähimmän naapurin menetelmään

Lähestymistavan heikkous on se, että usein $|L(G)| = \infty$

Symbolijonojen vertailuun tarvitaan jokin mitta, joka kuvaa niiden samankaltaisuutta

Yleensä esim. Euklidinen metriikka ei käy samankaltaisuuden mitaksi (vertailtavat symbolijonot voivat olla eri pituisia!)

Samankaltaisuusmitan tulee huomioida sekä symbolijonojen symboleiden samankaltaisuus että niiden sisäinen rakenne (esim. sanojen oikeinkirjoitus ja ääntäminen (ruotsi-ruatsi ja betoni-petonni) tai taivutus)

Samankaltaisuusmitan valinta riippuu luonnollisesti hyvin paljon hahmojen esitystavan (symbolit) valinnasta ja sovellutusongelmasta ominaispiirteistä (rakenteiden merkitys)

Esimerkki: käsinkirjoitetut merkit voidaan esittää symbolijonoina, joissa eri symbolit vastaavat vakiopituisia, mutta eri suuntaisia kaaren pätkiä (*'chain coding'*). Symboleiden vaihtuminen toisiksi voi merkitä, että jokin merkin osa on piirretty vähän eri suuntaan. Symboleiden puuttuminen tai lisääntyminen voi tarkoittaa, että jokin merkin osa on pienempi tai suurempi

Seuraavassa tarkastellaan aluksi yleisempää ongelmaa eli aikasarjojen vertailua

Dynamic Time Warping -algoritmi

Jos aikasarjan datapisteiden (piirrevektoreiden) välille on määritelty sovituskustannus, esim. Euklidinen etäisyys, voidaan käyttää **Dynamic Time Warping-algoritmia** (DTW):

- DTW-algoritmi sovittaa kahden aikasarjan datapisteet siten, että sovitettujen datapisteparien kustannuksien summa (tai tulo) on mahdollisimman pieni
- Olkoot $\mathbf{r}(i),\ i = 1, \ldots, I$, ja $\mathbf{t}(j),\ j = 1, \ldots, J$, kaksi aikasarjaa. Yleensä $I \neq J$
- Määritellään 2-ulotteinen avaruus, jonka piste (i, j) vastaa aikasarjojen datapisteitä $\mathbf{r}(i)$ ja $\mathbf{t}(j)$:
- Aikasarjojen sovitus voidaan esittää seuraavan polun avulla:

$$(i_0, j_0), (i_1, j_1), \ldots, (i_f, j_f) \tag{6}$$

- Jokaista polkua vastaa kokonaiskustannus D

$$D = \sum_{k=0}^{K-1} d(i_k, j_k), \text{ tai} \tag{7}$$

$$D = \sum_{k=0}^{K-1} d(i_k, j_k | i_{k-1}, j_{k-1}), \text{ tai} \tag{8}$$

$$D = \prod_{k=0}^{K-1} d(i_k, j_k | i_{k-1}, j_{k-1}), \tag{9}$$

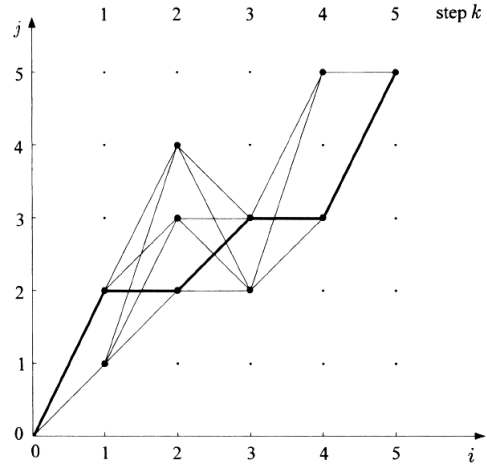
missä K on sovitettujen pisteparien lkm (polun pituus), ja $d(i_k, j_k)$ ja $d(i_k, j_k | i_{k-1}, j_{k-1})$ ovat datapisteiden $\mathbf{r}(i_k)$:n ja $\mathbf{t}(j_k)$:n väliset sovituskustannukset silloin, kun edelliset sovitukset joko ei huomioida tai huomioidaan

- Poluille voidaan asettaa erilaisia rajoituksia, esim. kuinka sovitetaan aikasarjojen ensimmäiset ja viimeiset pisteet, kuinka paljon polku voi poiketa lineaarisesta sovituksesta lokaalisti ja globaalisti jne

- DTW-algoritmin ratkaisu perustuu *Bellmannin optimaalisuus-periaatteeseen:*
  - Merkitään optimaalista polkua seuraavasti:
    $$(i_0, j_0) \rightarrow^{\mathsf{opt}} (i_f, j_f) \tag{10}$$
  - Mikäli optimaalinen polku kulkee pisteen (i, j) kautta, voidaan merkitä:
    $$(i_0, j_0) \rightarrow^{\mathsf{opt}}_{(i,j)} (i_f, j_f) \tag{11}$$
  - Bellmannin periaatteen mukaan
    $$(i_0, j_0) \rightarrow^{\mathsf{opt}}_{(i,j)} (i_f, j_f) = (i_0, j_0) \rightarrow^{\mathsf{opt}} (i, j) \oplus (i, j) \rightarrow^{\mathsf{opt}} (i_f, j_f), \tag{12}$$
    missä $\oplus$ tarkoittaa polkujen yhdistämistä
  - Edellisen perusteella optimaalinen polku voidaan ratkaista rekursiivisesti:
    $$D_{\mathsf{min}}(i_k, j_k) = \min_{(i_{k-1}, j_{k-1})} [D_{\mathsf{min}}(i_{k-1}, j_{k-1}) + d(i_k, j_k | i_{k-1}, j_{k-1})] \tag{13}$$

(jos polkua vastaava kokonaiskustannus on tulomuotoinen korvataan summa tulolla)

– Esimerkki rekursiivisesta ratkaisun etsimisestä:



- DTW-etäisyyttä on käytetty paljon esim. puheen ja käsinkirjoitetun tekstin tunnistuksessa

Yksi paljon käytetty DTW-algoritmiin perustuva mitta symbolijonojen vertailussa on **editointietäisyys:**

- Oletetaan, että hahmot koostuvat symbolijonoista (ja symbolien järjestyksellä on väliä!)
- Mitataan kahden symbolijonon, $\mathbf{A}(i),\ i = 1, \ldots, I$, ja $\mathbf{B}(j),\ j = 1, \ldots, J$, samankaltaisuutta tarvittavien editointioperaatioiden lkm:ien (symbolin vaihtoja C, lisäyksiä I ja poistoja R) avulla:

$$D(\mathbf{A}, \mathbf{B}) = \min_j [C(j) + I(j) + R(j)], \tag{14}$$

  missä j indeksoi kaikki mahdolliset muokkaukset, joilla $\mathbf{B}$:stä saadaan $\mathbf{A}$
- Määritellään 2-ulotteinen avaruus, jonka piste (i, j) vastaa symboleita $\mathbf{A}(i)$ ja $\mathbf{B}(j)$
- Etsitään optimaalista sovituspolkua pisteestä $(0, 0)$ pisteeseen (I, J)

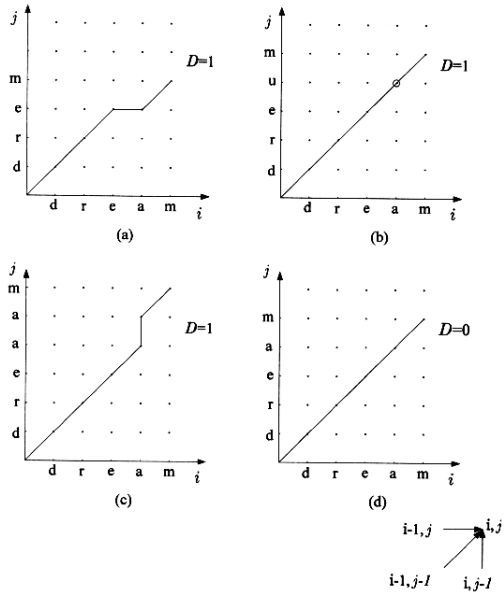
- Määritellään polun pisteelle $(0,0)$ kustannus $D(0,0) = 0$
- Polun pisteeseen (i,j) voidaan päästä vain pisteistä $(i-1,j)$, $(i-1,j-1)$ tai $(i,j-1)$
- Symbolien sovituksiin liitetään seuraavat kustannukset:

$$d(i,j|i-1,j-1) = \begin{cases} 0, \text{ jos } \mathbf{A}(i) = \mathbf{B}(j) \\ 1, \text{ jos } \mathbf{A}(i) \neq \mathbf{B}(j) \text{ (symbolin vaihto)} \end{cases} \tag{15}$$

$$d(i,j|i-1,j) = d(i,j|i,j-1) = 1 \text{ (symbolin poisto tai lisäys)} \tag{16}$$

- Optimaalinen sovituspolku voidaan nyt ratkaista rekursiivisesti

- Esimerkkejä:



Symbolijonojen vertailuun perustuva tunnistus on järkevää ainoastaan silloin, kun voidaan muodostaa ongelmaan hyvin sopiva sanakirja ja samankaltaisuuden mitta

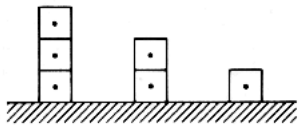
(symbolijonojen vertailun eräänlaisena laajennuksena voidaan pitää graafimuotoisina esitettyjen hahmojen vertailua)

1.4 Graafien vertailuun perustuva tunnistus

Suunnatun graafin avulla voidaan esittää monimutkaisempia piirteiden välisiä suhteita kuin yksiulotteisilla symbolijonoilla

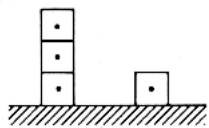
Ajatellaan, että graafin $G = \{N, R\}$ solmut N ovat (erilaisia) piirteitä ja niitä yhdistävät kaaret R kuvastavat (erilaisia) piirteiden välisiä suhteita (atribuuttigraafi)

Esimerkki graafeina esitetyistä hahmoista:



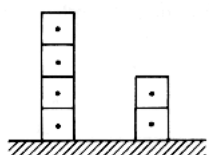
(a)

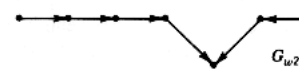
(b)



(c)

(d)





Tunnistus perustuu havaitun hahmon graafin ja eri luokkien prototyyppigraafien vertailuun ja esim. k:n lähimmän naapurin menetelmään

Havainnon graafi ei välttämättä vastaa täydellisesti mitään prototyyppigraafia. Havainnon graafi voi olla esim. vain jokin prototyypin osagraafi

Graafien välinen samankaltaisuusmitta perustuu sekä graafien solmujen että niitä yhdistävien kaarien samankaltaisuuteen

Graafit voidaan esittää $p \times p$-kokoisen vierekkäisyysmatriisin M ('adjacency matrix') avulla, missä p on solmujen lkm ja $M(i,j) \neq 0$, jos solmujen i ja j välillä on jonkinlainen kaari. Muuten $M(i,j) = 0$

Graafien homo- ja isomorfismi

Graafien $G_1 = \{N_1, R_1\}$ ja $G_2 = \{N_2, R_2\}$ sanotaan olevan *homomorfisia*, jos on olemassa kuvaus $f: N_1 \rightarrow N_2$ siten, että jos solmujen $v_1, w_1 \in N_1$ välillä on kaari graafissa G_1, niin silloin myös graafissa G_2 on (samanlainen)

kaari solmujen $f(v_1), f(w_1) \in N_2$ välillä eli

$$(v_1, w_1) \in R_1 \Rightarrow (f(v_1), f(w_1)) \in R_2 \tag{17}$$

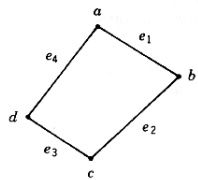
Graafit ovat *isomorfisia*, jos ne ovat homomorfisia ja kuvaus f on 1:1 ja kaikki N_2:n alkiot ovat jonkin N_1:n alkion kuvapisteitä (bijektio) eli

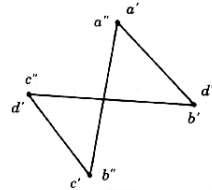
$$(v_1, w_1) \in R_1 \Leftrightarrow (f(v_1), f(w_1)) \in R_2 \tag{18}$$

(toisin sanoen graafista G_1 saadaan graafi G_2 indeksoimalla solmut uudestaan)

Subisomorifismi tarkoittaa sitä, että graafin G_1 osagraafi on isomorfinen jonkin graafin G_2 osagraafin kanssa

Esimerkki isomorfisista graafeista:





(Symmetric adj. matrix)

	a	b	c	d
a	0	1	0	1
b	1	0	1	0
c	0	1	0	1
d	1	0	1	0

	a′	b′	c′	d′
a′	0	1	1	0
b′	1	0	0	1
c′	1	0	0	1
d′	0	1	1	0

$f(a) = a''$
$f(b) = b''$
$f(c) = c''$
$f(d) = d''$

	a″	b″	c″	d″
a″	0	1	0	1
b″	1	0	1	0
c″	0	1	0	1
d″	1	0	1	0

Käytännön ongelmissa graafien isomorfisuus ei ole hyvä luokittelukriteeri, koska se ei salli graafeille pieniäkään poikkeamia eikä siksi siedä lainkaan virheitä esim. piirreirrotuksessa

Graafien isomorfisuuden toteaminen on lisäksi laskennallisesti hyvin raskasta.

Usein käytetäänkin testejä, jotka epäonnistuvat, jos graafit eivät ole isomorfisia. Seuraavat graafien ominaisuudet ovat invariantteja isomorfismin suhteen:

- (tietynlaisten) solmujen lkm
- (tietynlaisten) kaarien lkm
- (tietynlaisten) solmuun tulevien, solmusta lähtevien tai solmuun liittyvien, suuntaamattomien kaarien lkm:t
- l:n pituiset syklit

Graafien vertailuun on kehitetty useita erilaisia menetelmiä, jotka sallivat poikkeamat graafien välillä. Kaksi tyypillistä lähestymistapaa:

- Muodostetaan graafeille piirrevektorit ja vertaillaan niitä

- Käytetään graafien samankaltaisuuden mittana pienintä tarvittavien editointioperaatioden lkm:ää, joilla graafit saadaan muutettua toisikseen:
  - solmujen lisäys ja poisto
  - solmujen yhdistäminen ja jakaminen
  - solmujen tyypin vaihto
  - kaarien lisäys ja poisto
  - kaaren tyypin vaihto

Edellisten lähestymistapojen ongelmana on sopivien piirteiden ja metriikan valinta tai laskennallinen kompleksisuus