Sekvenssianalyysiä bioinformatiikkaan – Toteutusdokumentti

Rakenne

Ohjelma koostuu neljästä sekvenssianalyysialgoritmista, jotka kaikki käyttävät hyväkseen samantyylistä dynaamisen ohjelmoinnin periaatetta. Jokaisessa vertaillaan kahta sekvenssiä, joiden kohdistuksia (alignment) pisteytetään ja näistä valitaan pisteytykseltään paras. Kaikissa käytetään kohdistusmatriisia (alignment matrix), jonka jokainen elementti vastaa kahden merkin kohdistamista kekenään. Kohdistusmatriisissa pysty- tai vaakasuuntaan siirtyminen tarkoittaa, että kohdistuksessa on jommassa kummassa syötteessä *aukko* (gap), kun taas diagonaalisuuntaan siirtyminen tarkoittaa, että merkeillä on *vastaavuus* (match) tai hyväksytään epätäydellisyys, jossa merkille tehdään *muunnos* (mismatch).

Ohjelma käyttää yksinkertaista tekstipohjaista käyttöliittymää ja ottaa kaikki syötteensä tekstitiedostoista. Käyttäjä valitsee ohjelman aikana syötetiedoston, ajettavan algoritmin sekä mahdollisesti algoritmin pisteytyksen.

Longest Common Subsequence

LCS-algoritmi etsii kahdesta sekvenssin, joka on pisin kaikista mahdollisista sekvensseistä, jotka ovat molempien syötteiden alisekvenssejä. LCS:n pisteytyksessä vain siirrot ovat mahdollisia ja merkkimuunnoksille annetaan äärettömän negatiivinen painoarvo. Näin löydetyissä kohdistuksissa kaikki merkit ovat kohdistettu identtisien merkkien kanssa, eikä merkkimuunnoksia tapahdu. Tästä löydetään alisekvenssi helposti valitsemalla ne merkit, joiden kanssa on kohdistettuna toinen samanlainen merkki. Tämä sekvenssi on välttämättä näin molempien syötteiden alisekvenssi.

Global Sequence Alignment

GSA-algoritmi etsii parhaiten pisteytetyn kohdistuksen, jossa otetaan huomioon molempien syötteiden koko pituus. GSA-algoritmissa saa antaa toivotun pistemäärän sekä vastaavuudelle, muunnokselle, että aukolle. Kun GSA on täyttänyt kohdistusmatriisin, se aloittaa oikeasta alakulmasta ja etsii reitin, jota pitkin sinne on päästy vasemmasta yläkulmasta. Tämä reitti vastaa etsittyä kohdistusta.

Local Sequence Alignment

LSA-algoritmi toimii muuten kuten GSA, mutta se etsii kohdistuksen, jossa jotkin syötteiden osat ovat mahdollisimman hyvin kohdistettuna välittämättä muusta alueesta. Tämä tapahtuu siten, että kohdistusmatriisiin ei merkitä lainkaan negatiivisia pisteita, vaan kohdistus voidaan aloittaa mistä tahansa merkeistä. Kun kohdistusmatriisi on täytetty, LSA aloittaa koko matriisin suurimmasta pistemäärästä, ja etsii reitin jota pitkin sinne on kuljettu nollasta pisteestä. Tämä reitti vastaa haluttua kohdistusta.

Global Sequence Alignment with Gap Penalties

Käytännössä on usein parempi pisteyttää suuret aukot kohdistuksissa epälineaarisesti: mikäli sekvenssissä on mutaatio, se voi aivan yhtä hyvin olla suuri kuin pieni. Sen takia halutaan käyttää pisteytystä, jossa jokainen aukko saa alkaessaan suuren negatiivisen pistemäärän ja sen pituuden mukaan annettavan pienen negatiivisen pistemäärän. Käytännössä tämä tarkoittaa sitä, että yhden kohdistusmatriisin sijaan käytetään kolmea. Yksi pitää kirjaa pystysuuntaisista (ensimmäisen syötteen), toinen vaakasuuntaisista (toisen syötteen) aukoista ja kolmas pitää kirjaa ainoastaan diagonaalisiirtymistä, eli vastaavuuksista ja muunnoksista. Kun kaikki matriisit on täytetty, algoritmi etsii polun aivan kuten GSA:kin, mutta sallien siirtymät näiden kolmen matriisin välillä.

	-	A	G	A	T	G	A	C
-	0	-1	-2	-3	-4	-5	-6	-7
G	-1							
T	-2							
A	-3							
C	-4							

	-	A	G	A	C	G	A	C
-	0	-1	-2	-3	-4	-5	-6	-7
G	-1	-2	+4	+3	+2	+1	0	-1
T	-2	-3	+3					
A	-3							
C	-4							

Kuva 1. Esitäytetty kohdistusmatriisi skevensseillä GTAC ja AGATGAC, jossa muunnossakko on -3, aukkosakko -1 ja vastaavuusbonus +5 pistettä. Matriisia täytettäessä vertaillaan kolmea edeltävää alkiota. Havaitaan, että T ja G eivät vastaa toisiaan, joten diagonaalisiirtymästä tulisi -3 pisteen sakko. Pysty- ja vaakasiirtymästä tulee -1 pisteen sakko, joten paras pistemäärä saadaan kun valitaan pystysiirtymä. Tämä vastaa kohdistusta, jossa T kohdistetaan tyhjän merkin kanssa.

Suorituskyky

Kaikki algoritmit toimivat neliöllisessä ajassa. Ensin valmistellaan algoritmi ajettavaksi, eli luetaan tiedostolta syötteet koneen muistiin. Tämä ei vielä ole osa varsinaista algoritmia, joten se ei vaikuta aikavaativuuteenkaan. Tämän jälkeen valmistellaan pisteytysmatriisi, joka on aakkoston pituinen neliömatriisi. Aakkosto ei tietenkään voi olla pidempi kuin syöte, joten pisteytysmatriisin valmistelun aikavaativuus on $O(n^2)$. Käytännössä tämä on missä tahansa järkevässä syötteessä vakioaikainen toimenpide, sillä aakkosto on korkeintaan kymmeniä merkkejä kun taas syöte voi olla tuhansia. Tämän jälkeen valmistellaan kohdistusmatriisi, eli luodaan matriisi jonka sivujen pituudet vastaavat syötteiden sivujen pituuksia ja tarvittaessa alustetaan sen ensimmäinen rivi ja sarake valmiiksi. Kohdistusmatriisin valmistelun aikavaativuus on siis $O(n)$.

Kun valmistelut on tehty, tapahtuu varsinainen työ, eli matriisin pisteytys. Pisteyttäessä tutkitaan matriisin aikaisempia arvoja kolmesta alkiosta ja lasketaan paras pistemäärä riippuen siitä mikä operaatio tehdään seuraavaksi:

```
matriisi[i][j] = max( (matriisi[i][j-1] - δ),
          (matriisi[i-i][j] - δ), (matriisi[i-1][j-i] + μ))
```

Tässä δ on aukon pistemäärä ja μ on joko vastaavuuden positiivinen pistemäärä tai muunnoksen negatiivinen pistemäärä jotka löydetään vakioajassa pisteytysmatriisista. Kaikkien kolmen pistemäärän laskeminen on siis vakioaikaista, kuten on myös maksimin etsiminen. Yhden alkion pisteytyksen aikavaativuus on siis $O(1)$ ja näinollen koko matriisin täyttämisen $O(n^2)$. Mikäli aukot pisteytetään epälineaarisesti, tehdään hieman monimutkaisempi vertailu. Tällöin meillä on käytössä kolme kohdistusmatriisia, M, X ja Y. Täytetään matriisit X ja Y seuraavasti:

```
Y[i][j] = max( (matriisiY[i-1][j] – σ), (matriisi[i-1][j] - (ρ+σ)) )
X[i][j] = max( (matriisiX[i-1][j] – σ), (matriisi[i][j-1] - (ρ+σ)) ).
```

Tässä σ on aukon aloittamisesta seuraava negatiivinen pisteytys ja ρ on aukon jatkamisen negatiivinen pisteytys. Matriisien X ja Y täyttö riippuu siis vain yhdestä kyseisen matriisin elementistä sekä yhdestä matriisin M elementistä. Tämän jälkeen täytetään matriisi M siten että

```
M[i][j] = max( (M[i-1][j-1] +  δ), X[i][j], Y[i][j]).
```

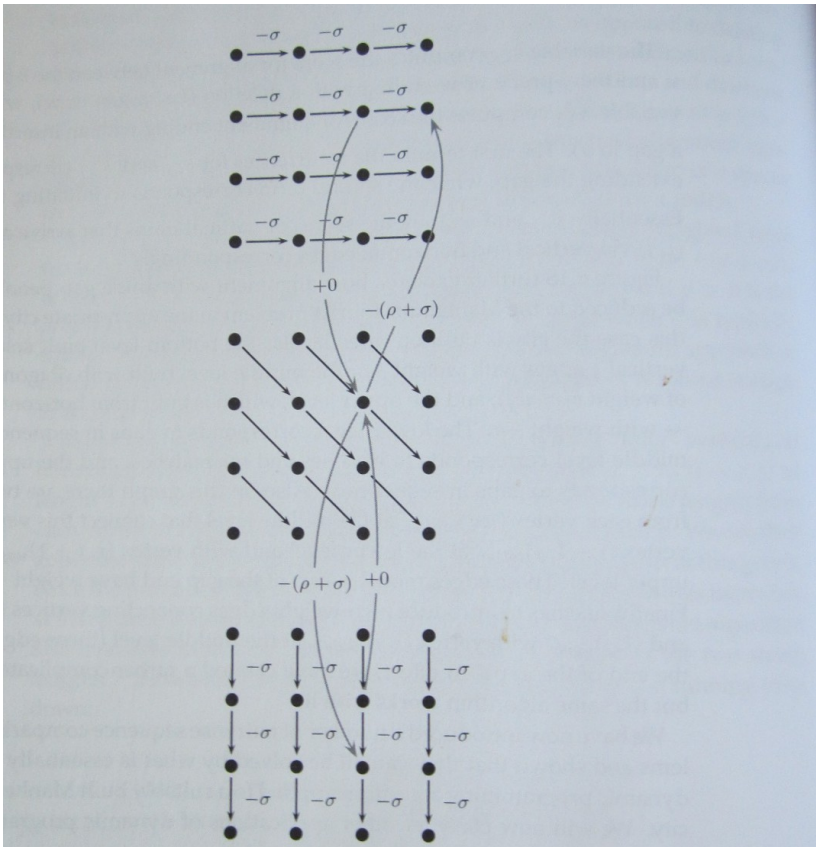

Figure 6.18 A three-level edit graph for alignment with affine gap penalties. Every vertex (i, j) in the middle level has one outgoing edge to the upper level, one outgoing edge to the lower level, and one incoming edge each from the upper and lower levels.

Kuva 2: Epälineaarisessa aukkojen pisteyksessä käytettävät kolme matriisia vastaavat kaikki eri suuntaisia siirtymiä. (Kuva Jones & Pevzner s.186)

Tehtävää on siis huomattavasti enemmän. Kuitenkin X[i][j]:n ja Y[i][j]:n, laskeminen ovat molemmat vakioaikaisia operaatioita, joten myös M[i][j] saadaan laskettua vakioajassa. Näinollen kaikki kolme matriisia saadaan täytettyä aikavaativuudella $O(n^2)$.

Kun kohdistusmatriisi on täytetty, jäljellä on ratkaisun etsiminen. Jokainen kohdistus vastaa yhtä reittiä kohdistusmatriisin yhdstä alkiosta toiseen, ja matriisissa voidaan liikkua vain yhteen suuntaan pysty-, vaaka- tai diagonaaliaskelin, joten reitin pituus on korkeintaan 2n. Koska kohdistusmatriisiin tallennetaan laskettaessa myös reittitieto, siirtymä alkiosta toiseen on vakioaikaista ja reitti saadaan laskettua ajassa O(n). Näinollen koko algoritmin yhteinen aikavaativuus on $O(n^2)$.

Tilavaativuus on yllämainitulla logiikalla myös $O(n^2)$ kaikissa algoritmeissa. On kuitenkin huomionarvoista, että epälineaarinen aukkopisteytys kasvattaa tilavaativuuden kolminkertaiseksi, joten vaikka se ei vaikutakaan O-notaatiolla esitettävään tilavaativuuteen, se voi käytännössä kuitenkin olla huomattavakin rajoite. Kuten testauksessa kävi ilmi, tämä tarkoittaa esimerkiksi sitä, että Javan oletusarvoinen heap size tulee vastaan huomattavasti aikaisemmin kuin muilla algoritmeilla.

Katse tulevaisuuteen

Ohjelma on rakennettu siten, että sitä olisi helppoa laajentaa. Kaikki algoritmit perivät perusominaisuutensa abstraktilta luokalta PSA (Pairwise Sequence Alignment), joka sisältää työkalut kahden sekvenssin tutkimiseen yllä kuvatuin menetelmin. Tästä luokasta voisi luoda aliluokkia, jotka toteuttaisivat muita samantyyppisiä algoritmeja. Luokan PSA voisi myös laajentaa sellaiseksi, että sillä voisi tutkia useampaa kuin kahta sekvenssiä (Multiple Sequence Alignment). Mikäli ohjelmaan haluaa laittaa eri tyyppisiä algoritmeja, tulee niiden toteuttaa rajapinta Problem, jolloin niitä voidaan käsitellä pääohjelmassa kuten jo toteutettuja algoritmeja.

Tällä hetkellä ohjelman käytettävyys on vähän kyseenalaista, sillä kyseessä on täysin tekstipohjainen käyttöliittymä, joka etenee valinta kerrallaan. Parempi toteutus olisi yksinkertainen graafinen käyttöliittymä, jossa kaikki valinnat voisi tehdä ensin ja naksauttaa kerran OK, jonka jälkeen ohjelma suorittaisi halutut toimenpiteet. Koska tähän ei tarvittaisi minkäänlaista dynaamisuutta, käyttöliittymän ohjelmointi olisi aika yksinkertaista.

Tavoitteena on alusta lähtien ollut laajennettava ohjelma, joka noudattaa hyviä ohjelmistosuunnittelun periaatteita. Täten tulos on modulaarinen, mutta ei kovin optimoitu. Tuntuu, että koodia on aika paljon verrattuna siihen kuinka yksinkertaisi algoritmit toisaalta ovat. Koko ohjelmaa olis varmasti mahdollista virtaviivaistaa ja näinollen kenties myös tehostaa. Tosin koska projekti on toteutettu Javalla, lienee hyvä ohjelmoida tavoilla, joilla Javan oliomaailman vahvuudet tulevat esille.

Lähteet

- "An Introduction to Bioinformatics Algorithms", Jones, Neil C. ja Pevzner, Pavel A. MIT Press, 2004.
- Settles, Burr: “Integrated Biological Summer Research Program: Computational Biology & Biostatics Workshop” (Course material), 2008. http://pages.cs.wisc.edu/~bsettles/ibs08/
- Altschul, Stephen F., “Computational Genomics” (Course material), https://www.cs.umd.edu/class/fall2011/cmsc858s/
- ICS 161: Design and Analysis of Algorithms, Lecture notes for February 29, 1996. https://www.ics.uci.edu/~eppstein/161/960229.html