Sekvenssianalyysiä bioinformatiikkaan - Testausdokumentti

Yksikkötestaus

Yksikkötestaus suoritettiin Junit-testeillä. Jokaiselle (ei-abstraktille) luokalle tehtiin omat testinsä. Apumatriisien testeissä (AlignmentMatrix ja ScoringMatrix) testattiin yksinkertaisilla syötteillä kaikkien metodien (paitsi triviaalien settereiden ja gettereiden) toimivuus. Algoritmeja testattiin kolmella sekvenssiparilla, triviaalilla, yksinkertaisella ja monimutkaisella. Testit kirjoittavat ensin sekvenssit väliaikaiseen tiedostoon, josta algoritmit lukevat ne aivan kuten normaalin ajon tapauksessakin. Tätä varten testit perivät tiedostonkirjoituksen yläluokasta TestFileOperations.

LCS-algoritmin tapauksessa testattiin, että algoritmi tuottaa oikean pistemäärän ja että tuotettu tulos todellakin on molempien syötesekvenssien alisekvenssi. GSA-algoritmin tapauksessa testattiin myös pistemäärä ja lisäksi varmistettiin, että algoritmin antama vastaus ja sen ilmoittama pistemäärä vastaavat toisiaan. Samoin toimittiin myös laajennetun GSA with Gap Penaly -algoritmin tapauksessa. LSA-algoritmin kanssa verrattiin algoritmin antamaa pistemäärää oikeaan pistemäärään ja optimaaliseen ratkaisuun.

Algoritmien tuottamien vastausten oikeellisuuden tarkastamisessa ongelmaksi muodostuu se, että vastaukset eivät ole (eivätkä pyrikään olemaan) yksikäsitteisiä. Samaan pistemäärään päästään useilla eri ratkaisuilla, joista suurin osa on biologisessa mielessä (lähes) yhtä hyviä. Niinpä testauksessa ollaan enemmän kiinnostuneita pisteistä ja siitä, täyttääkö annettu ratkaisu tietyt ehdot, kuin siitä onko se jokin tietty sekvenssi. Ongelma tulee erityisesti esiin LSA-algoritmin tapauksessa, jossa ei ole mitään keinoa tarkistaa, onko vastaus ”oikea”. Ainoa tapa on siis verrata tulosta tunnettuun tulokseen, joka joudutaan siis tuottamaan käsin laskemalla, jotta tiedetään tuloksen olevan juuri toivotun algoritmin tuottama tulos.. Muissa algoritmeissa pystyttiin käyttämään olemassaolevia algoritmien implementaatioita tulosten vertailuun.

Käyttämiäni luotettavaksi katsottuja algoritmien implementaatioita:

http://www.bioinformatics.org/sms2/pairwise_align_dna.html

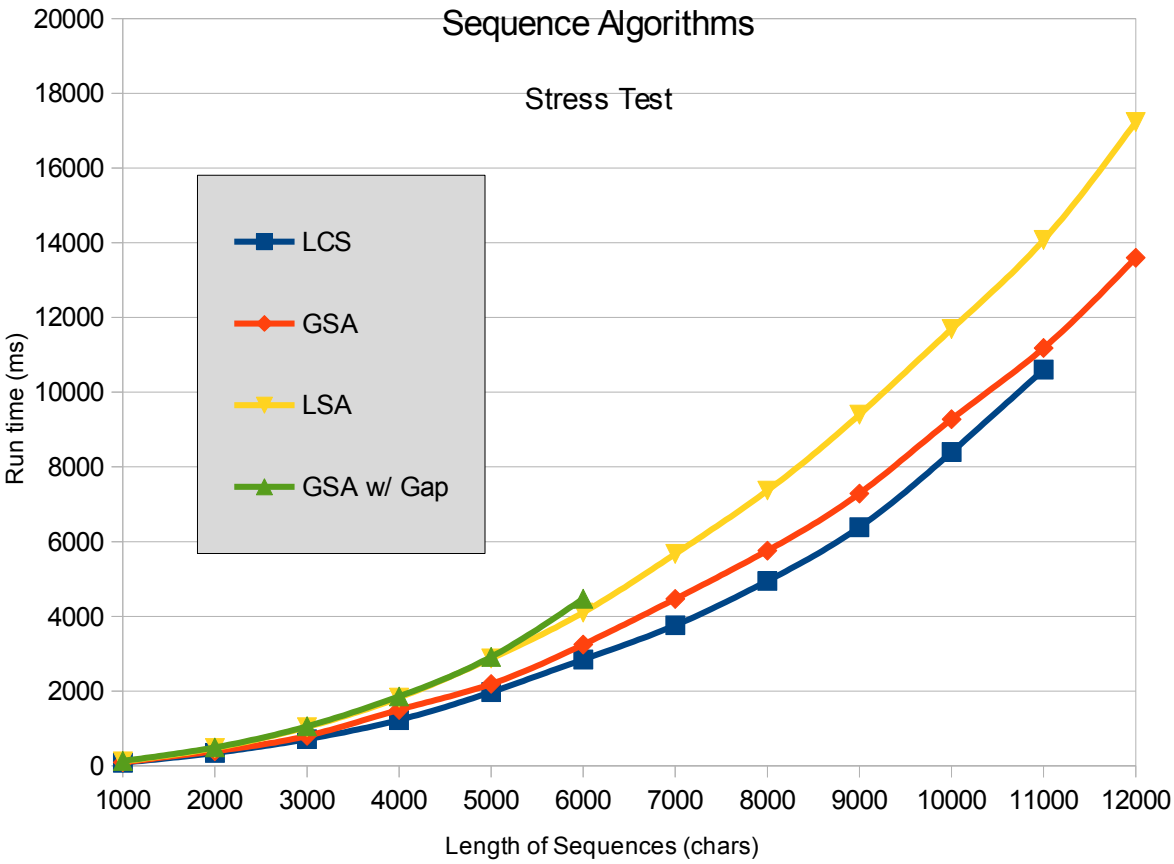
http://tandem.bu.edu/align.tool.html

https://www.ebi.ac.uk/Tools/psa/

Suorituskykytestaus

Rasitustestaus suoritettiin ajamalla algoritmit sarjalla syöttitä, joista jokaisessa oli kaksi satunnaista dna-sekvenssiä. Ensimmäisessä syötteessä oli kaksi sekvenssiä, joiden pituus oli 1000 merkkiä ja seuraavissa syötteissä sekvenssien pituus kasvoi aina tyhannella merkillä. Syötteitä pidennettiin kunnes JVM:n heap size limit tuli vastaan. Jokainen algoritmi ensin valmisteltiin ja sen jälkeen ratkaistiin kymmenen kertaa. Ratkaisuajoista laskettiin keskiarvo ja keskihajonta. Testit suoritettiin erikseen ajettavan ohjelman kautta, joka on ohjelman testipakkauksessa luokkana StressTest. Testit suoritettiin NetBeans-ympäristössä kotikoneellani, jossa testien aikana ei ollut muita ohjelmia käynnissä.

Suoritettaessa useita peräkkäisiä ajoja pullonkaulaksi muodostui Javan roskienkerääjän hitaus. Tämän vuoksi testatessa jokaisen testin välillä kannatti ajaa ”dummy-test”, eli jokin algoritmi lyhyellä syötteellä. Tämä (ilmeisesti) antoi roskienkerääjälle aikaa käydä tuhoamassa edellisen ajon Problem-olio vapauttaen sen käyttämän muistin. Mikäli näin ei tehty, testisarjat kaatuivat muistin puutteeseen jo 6000 merkin sarjoissa.



LCS-, GSA- ja LSA-algoritmit saatiin ajettua 12 kertaa, eli pisimmät syötteet olivat 12 000 merkin mittaisia. Joillakin ajoilla päästiin 13 kertaan, mutta tämä tuntui tapahtuvan täysin satunnaisesti enkä pystynyt keksimään mitkä tekijät asiaan vaikuttivat. GSA with Gap Penalty -algoritmi kaatui aina jo 6000 merkin jälkeen. Tämä on ymmärrettävää, sillä vaikka kyseinen algoritmi on tilavaativuudeltaan muiden tavoin neliöllinen, se kuitenkin käyttää muistia kolme kertaa enemmän kuin muut.

Keskihajonnat olivat pahimmillaan melko suuria, mikä näkyy myös siinä, että eri testiajoilla tulokset heittivät joskus huomattavastikin. Testejä ei kuitenkaan ajettu eristetyssä ympäristössä, joten taustalla tapahtuvat käyttöjärjestelmäprosessit vaikuttanevat asiaan.

Kuten graafista näkyy, suoritusajat noudattavat odotettua neliöllisyyttä. Algoritmien monimutkaisuuden lisääntyessä suoritusaikakin lisääntyy hieman, joten yksinkertaisin LCS-algoritmi on graafissa kauttaaltaan nopeampi kuin monimutkaisin LSA-algoritmi.