Puude teisenduskauguse arvutamine

Projekti raport aines "Algoritmika"

Raivo Laanemets

1 Probleemi kirjeldus

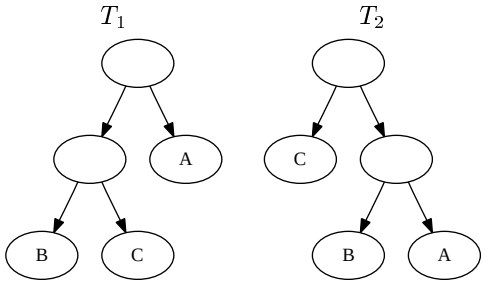
Projekti eesmärgiks oli uurida puude optimaalse teisenduskauguse probleemi. Kahe puu teisenduskaugus on operatsioonide arv või kogukaal, mille abil saab ühe puu teisendada teiseks puuks. Optimaalne teisenduskaugus on minimaalne kõikvõimalikest kaugustest või kogukaaludest.

Kui tegemist on märgendatud puudega, kus alamtippude järjestus on tähtis (*rooted-ordered tree*), siis võib teisendusteks lugeda tipu lisamise/kustutamise ja tipu ümbernimetamise. Selliste puude jaoks on teada algoritmid keerukusega $O(n^4)$ ning need kasutavad dünaamilist programmeerimist [1]. Projekti alguses sai üks sellistest algoritmidest realiseeritud.

Kui alamtippude järjestus ei ole tähtis (*rooted-unordered tree*), siis pole optimaalse teisenduskauguse lahendamiseks leitud veel polünomiaalset algoritmi, sest probleem on NP-täielik [2]. Projektis sai vaadatud kitsendatud varianti, kus on tegemist täielike kahendpuudega ja kus andmed asuvad ainult lehtedes ning ainsaks operatsiooniks on sisuliselt kahe suvalise alampuu ümbervahetamine. Tippude lisamist/kustutamist ei tehta, sest eeldatakse, et kahel võrreldaval puul on samasugused lehed. Sellisteks puudeks on näiteks fülogeneetilised puud [3]. Projektis sai põhiliselt uuritud teisenduskauguse leidmist just selliste puude jaoks.

1.1 Näide

Kahe alumise puu T_1 ja T_2 teisenduskaugus on 1. Alampuude järjestus pole oluline. Ükskõik kummas puus tuleb vahetada tipud A ja C.



2 Algoritm

Algoritm teisenduskauguse leidmiseks üritab ühendada lehtede paare, kus mõlemas võrreldavas puus on sama vanem. Sellise operatsiooni tulemuseks on uus leht, millega asendatakse ühendatud lehtede vanem. Kui selliseid paare ei leidu, tõstetakse lehti ümber, kuni selliseid paare leidub. Kuna kahe lehe ümbertõstmine võib mõjutada hilisemaid ümbertõstmiseid, siis tuleb leida kõik sellised ümbertõstmised ja valida neist "parimad". Algoritm lõpetab töö, kui mõlemas puus on alles jäänud ainult üks leht, mis algselt oli juur.

Et leida parimad ümbertõstmised, kasutatakse prioriteeditega magasini (kuhja), mille elementideks on puu teisenduste ahelad. Elemendid on järjestatud ahela kaalu järgi. Kui võtta ümbertõstmise kaaluks 1.0 ja ühendamise kaaluks 0.0, siis annab algoritm töö lõpus optimaalse teisenduskauguse, mis on võrdne kuhjas esimese ahela kaaluga, mis teisendab esialgsed puud ühe lehega puudeks.

2.1 Algoritmi üldskeem

Sisend: puud A ja B, alampuude ühendamise kaal w_1, ümbertõstmise kaal w_2.
Väljund: teisendusoperatsioonide jada c.

1. Prioriteetidega magasini Q initsialiseerimine.
2. Lisa magasini Q puu samasusteisendus (kaal 0.0).
3. Võta teisenduste jada c magasinist Q.
4. Leia puude A ja B lehed arvestades teisendusi jadas c.
5. Kui lehtede arv on 1, väljasta c ja lõpeta.
6. Leia ühendavad teisendused $t_1, \ldots, t_n$.
7. Lisa magasini Q c-st ja t_i-dest saadud jadad $c_i = c + t_i$ arvestades nende kaalusid $w(c_i) = w_1 + w(c)$, $i = 1, \ldots, n$.
8. Leia kõik lehti ümbervahetavad teisendused $t'_1, \ldots, t'_m$.
9. Lisa magasini Q c-st ja t'_i-dest saadud jadad $c'_i = c + t'_i$ arvestades nende kaalusid $w(c'_i) = w_2 + w(c)$, $i = 1, \ldots, m$.
10. Mine sammu 3.

3 Algoritmi realisatsioon

Programm, mis realiseerib kirjeldatud algoritmi, on kirjutatud keeles Java. Programm võtab sisendiks võrreldavate puude failinimed ja väljastab leitud sobiva teisenduste jada töö lõpus. Programmile saab käsurealt ette anda järgmised parameetrid:

- `-in1`: esimese puu failinimi.
- `-in2`: teise puu failinimi.
- `-ms`: maksimaalne iteratsioonide arv sammudes 3-10.
- `-r`: randomiseeritud ümbertõstmiste kasutamine. Kui on käsureal määratud, siis sammus 8 valitakse $r * n$ teisendust, kus n on hetkel vaadeldavate lehtede arv.
- `-uw`: lehtede ühendamise kaal (w_1).
- `-sw`: alampuude ümbervahetamise kaal (w_2).

Realisatsiooni kriitiline osa on sammus 4 tehtava tegevuse võimaldamine. Programm üritab hoida mälus puude teisendusi ja mitte täielikke puid, et mälu kokku hoida. Hetkel töötav realisatsioon kasutab primitiivset lähenemist ja seetõttu suudab keskmisel lauaarvutil läbi vaadata umbes 20000 puud sekundis.

Programmi saab käivitada skriptist `td.sh`, mis käivitab Java-keelse koodi koos vajalike teekidega. Programm sisaldab kokku umbes 1200 rida koodi. Enne kasutamist tuleb programm kompileerida käsuga `ant jar`.

4 Tulemused

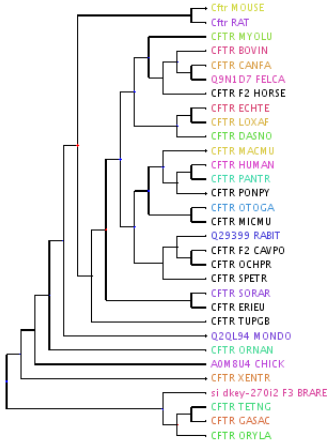
Algoritmi sai katsetatud fülogenetiliste puude võrdlemisel. Erinevate meetoditega genereeritud puud on mõnevõrra erinevad ja projekti raames realiseeritud algoritm annab puude sarnasuse teisenduskauguse abil. Programm väljastab ka puude ühildamiseks vajalikud sammud.

Programm loeb puud sisse NHX formaadis [4] faili seest, mis on lihtne ja levinud formaat bioloogiliste puude esitamiseks. Kuna erinevate meetodite saadud puud ei ole *TreeFam* lehelt otse saadaval, tuli alla tõmmata algkujul andmete kollektsioon (5GB) ja genereerida puud kasutades *TreeBest* vahendit [5].

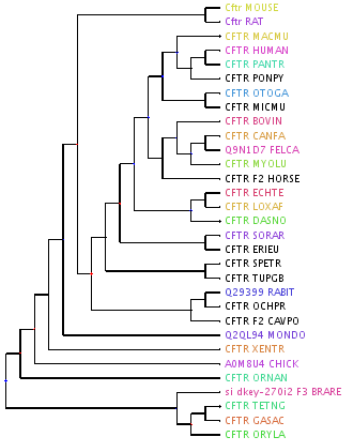
4.1 Näide sisendist

Sisendpuu T_1 on genereeritud käsuga
`treebest nj -t dm align/families_work/00/TF105200/full.nucl.mfa > tf105200.dm.nhx`.
Sisendpuu T_2 on genereeritud käsuga
`treebest nj -t ds align/families_work/00/TF105200/full.nucl.mfa > tf105200.ds.nhx`.

T_1



T_2



4.2 Näide väljundist

Programmi sisendiks sai antud ülalolevad puud. Optimaalset teisenduskaugust ei õnnestunud leida. Seetõttu tuli häälestada ühendamisoperatsiooni kaalu, et algoritm valiks ahnemalt väikeseid puid (rohkem ühendamisi ära tehtud). Lisaks tuli sisse lülitada randomiseeritud vahetuste geneerimine.

Programmi käivitamiseks vajalik käsk on järgmine:
`./td.sh -in1 data/tf105200.ds.nhx -in2 data/tf105200.dm.nhx -uw -0.5 -ms 100000 -r 1.5`.

```
Merge Cftr_MOUSE and Cftr_RAT in both trees into L0
Merge CFTR_HUMAN and CFTR_PANTR in both trees into L7
Merge L7 and CFTR_PONPY in both trees into L13
Merge CFTR_MACMU and L13 in both trees into L19
Merge CFTR_OTOGA and CFTR_MICMU in both trees into L25
```

```
Merge L19 and L25 in both trees into L30
Merge CFTR_ECHTE and CFTR_LOXAF in both trees into L36
Merge CFTR_TETNG and CFTR_GASAC in both trees into L1446
Merge CFTR_CANFA and Q9N1D7_FELCA in both trees into L1645
Merge L1446 and CFTR_ORYLA in both trees into L1651
Merge si_dkey-270i2_F3_BRARE and L1651 in both trees into L1670
Merge CFTR_SORAR and CFTR_ERIEU in both trees into L1680
Merge L36 and CFTR_DASNO in both trees into L1682
Swap L0 and CFTR_ORNAN in first tree
Swap L0 and CFTR_ORNAN in first tree
Merge L0 and L0 in both trees into L1871
Merge L1871 and Q2QL94_MONDO in both trees into L1872
Swap CFTR_ORNAN and CFTR_XENTR in first tree
Merge L1872 and CFTR_ORNAN in both trees into L2259
Merge L2259 and AOM8U4_CHICK in both trees into L2260
Merge L2260 and CFTR_XENTR in both trees into L2261
Merge L2261 and L1670 in both trees into L2262
Swaps: 3
```

Väljundis väljastatakse info tehtud sammude kohta. Väljundi viimasel real väljastatakse kokku tehtud alampuude ümbervahetuse arv, mis näitabki tegelikku teisenduskaugust. Antud väljundini jõudis programm umbes 10 sekundi töötamise järel.

5 Kokkuvõte

Projektis sai põhiliselt uuritud puude teisenduskaugust ja koostatud Java-keelne programm optimaalse teisenduskauguse leidmiseks fülogeneetiliste puude jaoks. Algoritmi realisatsioon on paraku aeglane ja suudab töötada kuni paarikümne tipuga puude peal.

Viited

[1] Serge Dulucq; Hélène Touzet. Analysis of tree edit distance algorithms. *In Proceedings of the 14th annual symposium on Combinatorial Pattern Matching.* 2003

[2] Horesh Yair; Mehr Ramit; Unger Ron. Designing an A* algorithm for calculating edit distance between rooted-unordered trees. *Journal of computational biology : a journal of computational molecular cell biology 2006;13(6):1165-76.* 2006

[3] Tree families database. http://www.treefam.org/, 20.05.2009

[4] Christian M. Zmasek. NHX formaadi spetsifikatsioon. http://phylosoft.org/NHX/, 20.05.2009

[5] TreeSoft: TreeBest. http://treesoft.sourceforge.net/treebest.shtml, 20.05.2009