TARTU ÜLIKOOL
Loodus- ja täppisteaduste valdkond
Arvutiteaduse instituut
Informaatika õppekava

Karel Paan

Haigustrajektooride sarnasuse hindamise metoodika

Bakalaureusetöö (9 EAP)

Juhendaja: Prof. Jaak Vilo

Tartu 2021

Haigustrajektooride sarnasuse hindamise metoodika

Lühikokkuvõte:

Terviseandmete laialdane digitaliseerimine on avanud ukse personaalmeditsiini. Leides sarnaste haigustrajektooridega patsiente, on võimalik ennustada tulevasi diagnoose, haiguseid ennetada või nende mõjusid leevendada. Kuna bioinformaatikas leidub sarnase ülesehitusega andmeid, kasutatakse antud töös ära sealt valdkonnast õpitut. Selleks uuritakse töös meetodeid, mida kasutatakse bioinformaatikas sarnaste makromolekulide leidmiseks. Käesoleva töö eesmärgiks on tuvastada parim meetod, et leida sarnaseid haigustrajektoore kindla hulga patsientide seast. Selleks koostatakse meetodi katsetamiseks skript, mis leiab generaatori poolt loodud patsientide diagnooside seast sarnaste trajektooridega klastreid. Meetodi täpsuse kinnitamiseks kontrollitakse genereerimisel kasutatud signaali olemasolu leitud trajektoorides. Uurimustöö lõpus kirjeldatakse parima leitud meetodi töökäiku, analüüsitakse tulemusi ning tuuakse välja võimalikud viisid edasiarenduseks.

Võtmesõnad:
dünaamiline programmeerimine, bioinformaatika, kliiniline trajektoor, pikim ühisjada

CERCS:
B110 - Bioinformaatika, meditsiiniinformaatika, biomatemaatika, biomeetrika

Methodology for evaluating similarity in health trajectories

Abstract:

The vast digitalization of health data has opened a door towards personalized medicine. By methodically searching for patients with similar health trajectories it is possible to predict diagnoses and prevent diseases or alleviate their effect. Prior knowledge from bioinformatics can be leveraged because the data shares a similar structure and problems with medical data. This thesis aims to find the best method for identifying similar health trajectories inside a certain set of patients. To test the method a script is made that uses input generated from patients' health data and returns clusters with similar trajectories. The generator's signal is used as an indicator of effectiveness of the method. At the end of the thesis, the implemented method is described with ideas on improving it.

Keywords:
dynamic programming, bioinformatics, clinical trajectory, longest common subsequence

CERCS:
B110 - Bioinformatics, medical informatics, biomathematics, biometrics

Sisukord

Sõnastik

BLAST programm, mis leiab sarnasuse piirkondi bioloogliste jadade vahel [1] 12, 23

FASTA programm, mis leiab sarnasuse piirkondi bioloogliste jadade vahel[1] 23

meioos raku paljumine, mille käigus kopeeritakse loodavatele rakkudele pool genoomist 24

puriin lämmastikualuse tüüp, kuhu kuuluvad DNA lämmastiku alustest A ja G. [2] 28

pürimidiin lämmastikualuse tüüp, kuhu kuuluvad DNA lämmastiku alustest C ja D [2] 28

RHK-10 (ICD-10) Rahvusvahelise haiguste ja tervisega seotud probleemide statistilise klassifikatsiooni kümnes versioon[2] 31

[1] FASTA. Wikipedia. https://en.wikipedia.org/wiki/FASTA (29.07.2021)

[2] Tervisestatistika ja terviseuuringute andmebaas. https://statistika.tai.ee/Resources/PX/Databases/Andmebaas/02Haigestumus/09Vigastused/VIGinfo.htm (22.07.2021)

Lühendid

DNA desoksüribonukleiinhape 12, 24, 25, 28

DP dünaamiline programmeerimine 13, 18–24, 26, 27, 37

GA täielik joondamine (ingl *global alignment*) 18, 19, 27, 32

HA heuristiline algoritm 23

IUPAC Rahvusvahelise Puhta ja Rakenduskeemia Liit 25

LA kohalik joondamine (ingl *local alignment*) 18, 19, 27, 31–33

MSA mitmene joondamine (ingl *multiple sequence alignment*) 12, 13, 19

NWA Needleman-Wunschi algoritm 24, 26, 27

OMOP järgitavate terviseandmete partnerlus (ingl *Observational Medical Outcomes Partnership*) 9, 10, 31, 34

PA järkjärguline joondamine (ingl *progressive alignment*) 19

PWA paaris joondamine (ingl *pairwise alignment*) 12, 13, 19

RNA ribonukleiinhape 12, 24

SA ülekattega joondamine (ingl *semiglobal alignment, overlap alignment*) 19

SWA Smith-Watermani algoritm 26, 27

1 Sissejuhatus

Terviseandmete kättesaadavus muutub läbi aastate aina paremaks ja näide sellest on Eesti patsiendiportaal Digilugu[3], kust on kättesaadavad patsiendi diagnoosid isegi siis, kui andmed pärinevad erinevatest asutustest. Mida rohkem on andmeid, seda rohkem on nendes võimalik järeldada, eeldades, et kvaliteet ei lange kvantiteediga. Bioinformaatika sündis vajadusest analüüsida bioloogilisi andmeid ja tõdemusest, et käsitsi seda teha võtaks liiga palju aega. Sama on juhtumas ka andmetega meditsiini valdkonnas, kus aina vähem piisab sellest, et arst ise leiab soovitud tulemuse, kuna arvutite abil on tulemus täpsem ning ekspert saab rohkem aega panustada oma tööle. Töö eesmärgiks on selgitada, kas ja milliste jada võrdlusmeetoditega on võimalik leida sarnaseid haigustrajektoore. Kuna bioinformaatika on sarnase arengukäigu juba läbi käinud, võetakse töö koostamisel sealt valdkonnast palju eeskuju.

Terviseandmete käsitlemisel on suureks probleemiks ananonüümsuse tagamine, seetõttu on ligipääs nendele andmetele osalises või täies mahus piiratud. Taolise olukorra lahendamiseks on võimalik appi võtta kunstlikud ehk sünteesitud andmed [3]. Kunstlikud andmed võimaldavad luua soovitud hulgas ja ajaraamis testandmeid, mis sisaldavad soovitud hulgal müra (ingl *data noise*) ning signaali. Lisaks on andmed vabalt kättesaadavad, sest nad kuuluvad nende loojale. Kui päris terviseandmete puhul puudub võimalus veenduda vastuse õigsuses, siis testandmetele saab looja ise määrata signaali, mida hiljem kontrollimiseks kasutada. Antud töös kasutatakse meetodite testimise jaoks A. Valdase uurimustöö [4] käigus valminud diagnooside trajektooride generaatori poolt loodud kunstandmeid.

Töö koosneb neljast sisupeatükist (peatükid 2-5), millest esimeses räägitakse meditsiiniandmete omapärast ning varasematest analüüsimismeetoditest. Teises peatükis selgitatakse, kuidas on võimalik erinevaid jadasi omavahel võrrelda ja tuuakse näiteid arvutiteadusest ning bioinformaatikast. See peatükk on kõige sisukam, sest selle alusel on koostatud ka töö praktiline pool. Kolmandas sisupeatükis tutvustatakse valminud praktilist osa. Viimases peatükis analüüsitakse eelmises peatükis saadud tulemusi.

Töös kasutatud haigustrajektoorid, edaspid ka diagnooside trajektoor või lihtsalt trajektoor, tähendab antud töö kontekstis ühele isikule määratud diagnooside jada. Mõisteid sõne, sõna, jada, järjestus, seriaal kui ka seriaalsed andmeid, kasutatakse töös võrdväärsetena. Varasemalt on Tartu Ülikooli Arvutiteaduse instituudis A.-O. Mäesalu uurinud haiguste komorbiitsust Eesti populatsioonis, kasutades Eesti E-tervise Sihtasutuse 2012.-2013. aasta epikriiside andmeid. Matemaatika ja statistika instituudis on B. Rebane uurinud rekurrentseid tehisnärvivõrke inimeste kliiniliste trajektooride ennustamisel.

[3] Patsiendiportaal. `https://www.digilugu.ee` (18.07.2021)

2 Meditsiini andmed

Inimeste ravimisel on pikk ajalugu, esimesed teadaolevad leiud pärinevad juba Vana-Egiptusest. Lisaks ravimisele hakati üles märkima ravimiseks kasutatud aineid ning viise, et teadmisi talletada ja põlvkonniti edasi anda. Lähiajaloos hakati kirjeldama ka patsientide kohta käivaid andmed nagu vanus, sugu, diagnoosid jne, kuna kõik ravimid ja meetodid ei toimi üheselt kõikidele organismidele. Näiteks võib ravimil olla kõrvaltoime, mis on ohtlik südamele, seega antud ravim pole soovitatav südamehaigetele vaatamata, et toimeaine ei pruugi olla südamega seotudki. Tänu terviseandmetele on võimalik arvestada iga patsiendi individuaalsete vajadustega, kuid füüsilisi andmeid on väga raske jagada erinevate meditsiini asutuste vahel, samuti on neid raske hoiustada ja nendest informatsiooni leida. Selle probleemi lahendamiseks on näiteks haiglad hakanud rakendama tervise infosüsteemi (ingl *health information system*), mis on asutuse keskeks andmebaasiks meditsiini, k.a. terviseandmete hoidmiseks, lisamiseks ning nende otsimiseks. Veel enam pakub see uusi võimalusi patsientide põhiliste andmeanalüüside tegemiseks, sest käsitsi oleks nende tegemine liiga kulukas. Tulenevalt terviseandmete suurest parameetrite hulgast, pole nende analüüsimine olnud alati mõistliku aja piires võimalik, kuid tänu arvutite võimekuse rutulisele kasvule on see muutunud piisavat kiireks, mis omakorda on tekitanud rohkem huvi terviseandmete analüüsimise vastu. Antud peatükk vaatab, millised on terviseandmete omadused, kus on murekohad ning millised on hetkel kasutusel olevad meetodid nende analüüsimiseks.

2.1 Kirjeldus

Käesolevas töös loetakse meditsiiniandmete hulka diagnooside jadad ehk trajektoorid. Töös käsitletakse mõisteid haigustrajektoorid, terviseandmed ja meditsiiniandmeid võrdsetena. Definitsioon jäetakse võimalikult vabaks, kuna leidub erinevaid standardeid ning viise terviseandmete ülesmärkimiseks ning töö autor ei oma otsest eelistust mitte ühegi suhtes. Küll aga kasutatakse näidete jaoks OMOP andmemudelit ja kontseptsioone, kuna tegu on ühe populaarseima terviseandmete mudeliga, mis seob endaga teisi terviseandmete standardeid. Definitsiooni alla kuuluvad näiteks määratud ravimid, arsti visiidid, sümptomid, diagnoosid ning vaktsiinid.

2.2 Omadused

Selleks, et saaks töötada kindla andmeliigiga, on esmalt vaja aru saada, millised on tema omadused. Omaduste puhul ei pruugi alati selgelt eristuda, kas ta on hea, halb või mõlemat, sest kõik oleneb kontekstist. Kuna meditsiini valdkond jaguneb paljudeks eraldiseisvateks harudeks, millel on tugevalt välja kujunenud oma tavad ja distsipliinid, on raske piiritleda ühte kindlat konteksti. Antud peatükis tuuakse välja peamised omadused, mis mõjutavad meditsiiniandmete analüüsi tulemusi.

Tervise infosüsteemide areng algas seitsmekümnendatel [4]. See umbes viiekümne aastane periood on väiksem, kui tänapäeva keskmine eeldatav eluiga, lisaks ei kasutanud kõik haiglad koheselt infosüsteeme. Seetõttu pole enamus inimeste digitaalne terviseandmete kogumik täielik. Enne digitaliseerimist hoiti andmeid füüsilisel kujul, kuid neid digitaliseerides võis palju andmeid kaotsi minna või neid sisestati valesti. Ka tänapäeval, kasutades digitaalseid süsteeme, võib juhtumite ülesmärkimisel esineda vigu. Veel enam, kõik meditsiiniasutused ei pruugi omavahel koostööd teha ja andmeid jagada, samuti võib probleem olla ka erinevates andme standardites institutsioonide vahel, mis samuti takistab andmete ühildamist. Lisaks mängivad rolli juriidilised aspektid, näiteks piiratakse avaldada inimese täielik sünniaeg, et kaitsa inimese identiteeti. Seega tuleb andmete juures arvestada, et nad pole täiesti täpsed, sest võivad sisaldada vigu ning ei pruugi olla esindatud terves mahus. Ühtlasi on probleemiks see, et ei ole võimalik kindlalt teada saada, kui suures mahus esineb andmetes puudujääke.

Raske oleks kokku lugeda kindla sümptomi esinemiskordi, kui need oleks üles märgitud sõnaliselt, kuna haigusel võib mitu erinevat nime olla, võib esineda kirjavigu ja lisa tähemärke kirjelduses. Selliste vigade vältimiseks kasutatakse standardeid, nagu OMOP andmemudeli standardne sõnastik, kus igale sündmusele lisatakse oma unikaalne indeks, mille järgi on neid lihtne hiljem otsida. Sellegipoolest leidub haruldasi juhtumeid, mille jaoks puudub standardis kirje, see aga toob kaasa eelnevalt mainitud probleemid. Samuti on olemas ka erinevaid sõnastikke, mis kirjeldavad samu andmeid, näiteks haigusi ja see on lisaks üks probleemidest, mida standardne sõnastik lahendab, sest ta omab viiteid teiste sõnastike samaväärsetele indeksitele. Seega leidub hulk andmeid, näiteks haigusjuhud, mis on hästi dokumenteeritud, kuid sellegi poolest pole see lahendus täiuslik, sest väiksema tõenäosusega sündmusi ei pruugi olla standardselt kirjeldatud.

Meditsiini analüüsimiseks kasutatakse erinevate väljunditega aparaate. Röntgenipilt, vererõhu mõõt ning südame löögisageduse graafik on kolm erinevat andmetüüpi, mida omavahel ei saa ilma töötlemata võrrelda. Selleks, et terviseandmeid tervikuna analüüsida, peab saama neid omavahel võrrelda. Üks variant selleks oleks ebatavapäraste andmete muutmine numbriliseks või kategoorilisteks. Röntgenipildi saab asendada väärtusega 1, kui patsiendil on haigus x ja väärtusega 0 kui ei ole haigust x. See ei ole ideaalne lahendus mass andmete analüüsimiseks, sest vajaks käsitsi või masinõppega andmete töötlemist ning kuna võimalikke variante on palju, eeldab see, et uurijal on spetsiifilisem küsimus, millel vastust otsitakse.

Samu andmeid on võimalik ka erinevalt interpreteerida– ühes meditsiini asutuses on oluline koguda andmeid selle kohta, kas patsient suitsetab, teise meditsiiniasutuse jaoks aga on oluline teada täpset kogust. Seega poleks need andmed omavahel võrdväärsed, küll aga saaks hetkeolukorra näitel teise asutuse andmed teisendada kategoorilisele kujul,

[4]The History of Healthcare Technology and the Evolution of EHR - VertitechIT. https://www.vertitechit.com/history-healthcare-technology (05.01.2021)

mis esimese asutuse jaoks oleks piisav. Lisaks langevad meditsiini andmed erinevatesse kategooriatesse, nagu ravimite võtt, vastuvõtud, sümptomid, diagnoosid. On arusaadav, et need andmetüübid ei ole võrdse kaaluga, sest diagnoos on oluliselt kindlam indikaator, kui vastuvõtt, kuna vastuvõtu puhul ei pruugi olla kindel, millised olid kaebused. Veel enam on kaebused sõnalises vormis, seega ei ole kahte sarnase vastuvõtu põhjusega inimest võimalik tuvastada. Sellegipoolest ei ole aga teada, kui suure kaaluga mingi kindel andmetüüp peaks olema.

2.3 Analüüsi meetodid

Järgnev lõik tugineb OHDSI raamatule [5]. Enamus tänapäeval kasutuses olevad kliinilised ennustusmudelid on koostatud kasutades väikseid andmehulki, see aga tähendab, et andmete kasutaja peab tulemuste kasutamisel suurenisti lähtuma oma ekspertiisist. Nüüdseks on terviseandmete hulk oluliselt suurenenud, muutes probleemi vastupidiseks—andmete hulk on liiga suur, et seda oleks võimalik mõistliku aja jooksu käsitsi analüüsida. Selliseid andmeid nimetatakse Elektroonilisteks Tervisekirjeteks (ingl *Electronic Health Records*): sinna alla kuuluvad diagnoosid, ravimid, labori uuringute tulemused ja kliinilistest narratiividest pärinevad andmed. Elektrooniliste Tervisekirjete täielik potentsiaal ei ole veel teada. Selliste suurte andmekogumite analüüsimine on muutunud reaalsuseks tänu OMOP andmemudelile, mille abil saab meetodeid kinnitada erinevate osapoolte vahel ning samu meetodeid rakendada ilma ümbertöötluseta. OHDSI patsiendi tasemel ennustuste puhul uuritakse patsientide kogumit ning vaadeldakse mitmel patsiendil esineb mingi kindel tulemus (diagnoos, sümptom, vms) kindla perioodi jooksul alates vaatlusperioodist. Taoline lahendus otsib vastust küsimusele: “Uuritava grupi patsientide hulgast, kellel esineb tulemus uuritava aja piires?"[5]. Selline küsimuse püstitus töötab kõigi meditsiiniliste ennustuste puhul. Mudeli koostamiseks kasutatakse juhendatud masinõpet (ingl *superwised learning*), mis järeldab tulemuse kindlate karakteristikute (ingl *features*) ning märgendite (ingl *labels*) vahel. Märgendi koostamiseks otsitakse kõik juhud, millel eksisteerib kirje andmebaasis otsitava tulemuse jaoks. Masinõppe rakendamiseks on kasutusel hulk erinevaid algoritme ja mitu sellepärast, et erinevate andmete jaoks ei ole ühte algoritmi, mis oleks alati parim. Mõned näited kasutatavatest algoritmidest on: lassoregularisatsioon (ingl *lasso reularization*), otsustusmets (ingl *random forest*) ja k- lähima naabri meetod (ingl *k- nearest neighbors*).

3 Metoodika

Järgnevas peatükis kirjeldatakse hüpoteesi tõestamisel kasutatavaid meetodeid ning nende seost teemaga. Näited on peamiselt pärit bioinformaatika valdkonnast, kus võrreldakse omavahel valke, desoksüribonukleiinhapet (DNA) ja ribonukleiinhapet (RNA). Haiguste võrdlemisel tulevad kasuks samad meetodid, kuna ükski eelnimetatud objekt ei oma morfoloogilist tähendust. Morfoloogiline tähendus võib rolli mängida naturaalse keeletöötluse puhul, kus on jadal veel omaette tähendus. Näiteks, kui on kolm sõna: virn, vili ja pirn, siis kirjapildi poolest on vili ja pirn üksteisele sarnasemad, kuid tähenduse poolest on vili ja pirn üksteisega sarnasemad.

3.1 Võrreldava hulga valik

Kuna bioinformaatikas on enim levinud järjestikused andmed (täpsemalt peatükis 3.7.1), kasutatakse kahe või enam andme hulga võrdluseks joondamist. Enne joondamise algoritmi valimist on analüüsijal vaja teha oluline valik, kas keskendutakse kahe jada omavahelisele või mitme jada võrdlusele. Selle jaoks kasutatakse vastavalt paaris joondamist (PWA) või mitmest joondamist. Mõlemal on oma head ja vead, millega tuleks otsust langetades arvestada. PWA-d kasutatakse, et hinnata homoloogiat kahe jada vahel, MSA-d kasutatakse samal eesmärgil, kuid mitme jada puhul [6]. PWA jaguneb nii kohalikuks kui ka täielikuks, kuid MSA on enamasti täielik[5]. MSA kasutatakse ka mitme sarnase jada omavahel võrdlemiseks, mis paaris joondusega võtaks oluliselt rohkem aega [6].

Järgnev lõik toetub Claverie jt õpikule [7], kus tuuakse välja, et MSA puhul on pigem olulisem aeg, seega kasutatakse algoritme ja programme, mis on optimiseeritud kiirusele, mitte võimalikult suurele täpsusele. Peale uurijale huvipäraste järjestuste leidmist rakendatakse paaris joondamist, et kinnitada leitud järjestuste olulisus uurimiseks. Oluline on märkida, et mitmene joondus hõlmab endas ka paaris joondamist- näiteks BLAST otsib läbi andmebaaside paaris joondusega. Siin tulevadki mängu algoritmi omapärad.

3.1.1 Paaris joondamine

PWA on väga kasulik konserveerunud piirkondade leidmiseks [8]. Konserveerunud piirkond on evolutsiooni käigus sarnaseks jäänud ala ehk ala sees on toimunud vähe muutusi.

Järgnev lõik tugineb Claverie jt õpikul [7], kelle sõnul kasutatakse bioloogias tavaliselt PWA-d andmebaasi otsingu tulemuste korraldamiseks ning detailsete uuringute läbiviimiseks. Hea tava on andmebaasist leitud soovitud tulemused uuesti läbi joondada

[5]Difference between Pairwise and Multiple Sequence Alignment - Major Differences `https://www.majordifferences.com/2016/05/difference-between-pairwise-and-multiple-sequence-alignment.html#.YENfHGgzaiM` (06.03.2021)

kasutades PWA-d, et teha kindlaks, kas tegu on tõepoolest huvi pakkuva järjestusega. PWA abil tehakse kindlaks, kas kaks jada jagavad ühiseid omadusi või ühist domeeni, samuti on võimalik veenduda, kas kaks jada on tõepoolest omavahel homoloogsed. PWA-d tuleks kasutada ainult siis, kui otsitavate jadade vahel on suur tõenäosus leida homoloogia.

3.1.2 Mitmene joondamine

Kui kõiki sarnaseid jadasi saab võrrelda ühes tabelis, on mõistlik kasutada MSA-d [6]. MSA idee on täita tabel nii, et sarnased omadused oleksid samas veerus [7]. MSA on oluline tööriist leidmaks suhteid mitmete proteiini jadade vahel, kasulik ka fülogeenias, kus jada homoloogia abil määratakse liikide vahelisi suhteid [8]. Ükski tänapäeval leiduv MSA meetod ei ole täpne, kuna kõigis kasutatakse ligikaudseid väärtusi [7]. MSA arvestab kahe tähtsa omadusega mitme jada skoori hindamisel– mõned asukohad on rohkem konserveerunud kui teised (positsioonipõhine hindamine) ning jadad ei ole iseseisvad, vaid nad on seotud läbi fülogeneesi puu [6]. Enamus MSA-si kasutavad skoorimisel vahe karistust, kus kestva vahe pikendamine on vähem kulukam kui uue avamine [6].

3.2 Pikim ühis alamjada

Bioloogiliste sarnasuste otsimiseks leitakse kahe või enam jada vahel pikim ühine alamjada (ingl *longest common subsequence*). Järjestuse joondamine on pikima ühisjada alamliigiks [9]. See omakorda kuulub nii dünaamilise programmeerimise (DP) kui ka keeletehnoloogia valdkonda. Bioinformaatikas kasutatakse pigem alamjada, kuna andmed sisaldavad ühiseid domeene ehk motiive [8], mis on korduvad, kuid mitte alati üksteisele järgnevad, sisaldades lünki ning sellega aitab arvestada vahe (peatükk 3.2.1.1).

3.2.1 Teisenduskaugus

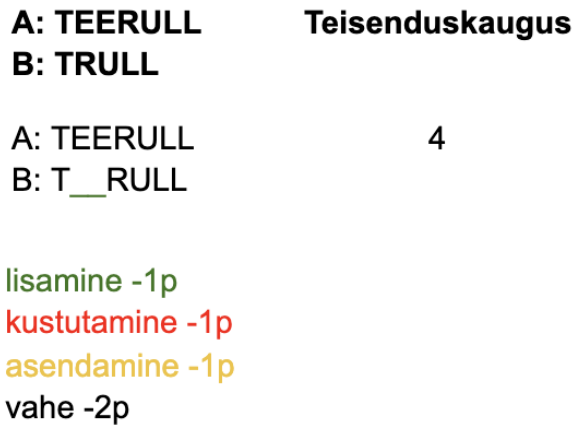
Võrdlemaks omavahel erinevate jada paaride analüüse on vaja näitajat, mis väljendaks, kui sarnased on kaks jada teineteisele. Üks taolisi näitajaid on teisenduskaugus, mis näitab, kui erinevad kaks sõne teineteise suhtes on. Selleks arvutatakse minimaalne operatsioonide arv, mis on vajalik ühe sõne muutmiseks teiseks. Tavapärased operatsioonid on muutmine ehk asendamine, mille käigus üks element asendatakse teisega, eemaldamine, mille käigus element eemaldatakse sõnest ning lisamine, mille käigus element lisatakse sõnesse. Leidub ka variatsioone, kus operatsioonina käsitletakse kahe kõrvuti oleva tähe vahetamist [10]. Teisenduskauguse liike on erinevaid, näiteks Hammingu kaugus, mis leiab väikseima asenduste arvu kahe sõne vahel, kus lubatud operatsiooniks on asendamine. Seega leiab Hammingu kaugus mitu elementi on kahe sama pikkuse

sõne vahel erinevad. Selline lahendus aga ei sobi bioloogiliste andmete jaoks, kuna võrreldavad jadad pole enamasti sama pikad.

Kõige levinum implementatsioon teisenduskauguse arvutamiseks on Levenshteini kaugus, mida seondatakse tihti teisenduskauguse endaga. Levenshteini kaugus leiab vähima operatsioonide arvu ühe sõne viimiseks teise kujule, kus sõnede pikkused võivad erineda ning operatsioonideks on lisamine, eemaldamine ning asendamine. Kuna nüüd on lubatud ka lisamine ja kustutamine, on võimalik kaugust arvutada kahe erineva pikkusega sõne vahel. Levenshteini kauguse arvutamise implementatsioonid võivad kasutada erinevate operatsioonide jaoks erinevaid kaale (karistusi), tavapäraselt omistatakse kõigile operatsioonidele sama kaal ning vaste puhul on kaal 0. Seega, kui joondada kaks identset sõna, oleks joonduse skoor 0. Asendamist karistatakse tihti ka -2 punktiga, kuna sellest võib mõelda kui kustutamisest ja lisamisest. Erinevate kaalude kasutamine võib olla kasulikum, kui sellega saab väljendada jadade konteksti. Ehk mitte soositud operatsioonile tuleks anda kõrgem karistus. Joonisel 1 on näidatud erinevaid viise, kuidas saab teisenduskaugust arvutada sõnade "TEERULL"ning "TRULL"vahel.

Joonis 1. Teisenduskauguse arvutamise võimalused.

3.2.1.1 Vahe



Joonis 2. Teisenduskauguse arvutamise võimalused vahega.

Teisenduskauguse arvutamisel on võimalik lisada ka vahe (ingl *gap*) operatsioon, mis mitmeküllastab võimalusi. Vahet saab kasutada olukorras, kus lisamine ja kustutamine on samaväärsed ning seega ei vaja täpsustamist. Samuti võib vahet kasutada koos eelmises peatükis mainitud operatsioonidega ning see on mõistlik olukorras, kus vahe esinemise asukohast vasakule ja paremale jäävad alamsõned rohkemate vastetega kui ilma vaheta, nagu on näidatud joonisel 2. Vahe konseptsioon on eriti väärtuslik bioloogiliste jadade joondamisel, kus esineb palju lühikesi inertsioone ja deletsioone homoloogsete jadade vahel. Neid võib teisisõnu nimetada ka müraks, seega kasutades vahesi, antakse sarnastele piirkondadele suurem väärtus üle mittesarnastele.

3.3 Järjestuste joondamine

Järgnevas alapeatükis käsitletakse joonduse leidmise rakendusi päriselus, tuues välja kasutatumad liigid. Kõiki liike kasutatakse nii DNA-s, RNA-s kui ka valkudes sarnasuste leidmiseks (bioloogilise funktsiooni või evolutsiooni perspektiivist lähtudes), omavahel võrdlemiseks. Liike on mitu, sest ükski ei ole universaalne. Selgitamaks, kas kaks valku on üksteisega samaväärsed, sobib täielik joondamine, aga tihti peale on vaja leida väikeseid sarnaseid mustreid kahe või mitme valgu vahel, mille jaoks sobib kohalik joondamine, mis selle probleemiga paremini toime tuleb. Peamised muutujad, millega valiku langetaja peab arvestama on: kiirus, kattuvus (kui suur osa jadast), kas tegu on võrdlemise või otsimisega ning täpsus. Näited algoritmide rakendamisest tuuakse peatükis **3.7.3** ning alampeatükkides.

3.3.1 Bioloogilised järjestused

Bioinformaatika peamiseks uurimisvaldkonnaks on molekulaartasand, millest tänapäeval on populaarsemad genoomid ning valgud [11]. Genoom koosneb DNA-st, mis omakorda koosneb neljast nukleotiidist, sarnaselt DNA-le koosneb valk 20-st aminohappest. Nii DNA, kui ka valgu elemendid moodustavad omavahel ahela, mida saab üles kirjutada jadana. Jadade kaudu on näiteks võimalik leida, kas kaks geeni või valku on homoloogilised ehk sarnased teineteisele [7]. Järjestuste joondamist kasutatakse olukorras, kus uuritakse andmete muutumist aja jooksul [6].

Järgnev lõik põhineb Remmi õpikul [2], kus kirjutatakse, et homoloogiliste piirkondade leidmine on üheks suurimaks analüüsi valdkonnaks. Teades homoloogseid piirkondi, saame hinnata järjestuse evulutsioonilist kaugust ning teada millised piirkonnad võiksid kanda sama funktsiooni. Läbi aja järjestused muutuvad, mille käigus võib täht muutuda (asendus), lisanduda (inertsioon) või kaduda (deletsioon). Enamasti osutuvad muudatused liiga massiivseks, et neid oleks võimalik silmaga võrrelda. Seepärast kasutatakse ühe või mitme järejstuse omavahel võrdlemiskes järjestuste joondamist.

Käesolev lõik põhineb Pisanti artiklil [12], kus seletatakse, et järjestuse joondamine on protsess, mille käigus võrreldakse ja leitakse sarnasusi (bioloogilises) jadas. Milliseid sarnasusi otsitakse, sõltub joondamsie eesmärgist. Kõige lihtsam viis kahe sama pikkusega jada võrdlemiseks on kokku arvutada mõlemas korduvad sümbolid. Väärtust, mis seda iseloomustab , kutsutakse joonduse tulemuseks, vastandväärtus, mis näitab kui palju kaks jada üksteisest erinevad, kutsutakse jadade vaheliseks kauguseks. Ei piisa vaid jada elementide asukohtade võrdlemisest, oluline on ka lisamine ja eemaldamine, kuna need on bioloogias toimuvad nähtused. Kui räägitakse jada joondamisest, siis tavaliselt peetakse silmas muutmise kaugust. Muutmise kaugus on jada, mis sisaldab võimalikult vähe muutmisi, lisamisi ja asendusi, et moodustada kahest jadast pikim ühisjada.

3.3.1.1 Vahekaristus

Käesolev lõik tugineb Gosh jt õpikul [6], kus kirjutatakse, et joondus võib anda parema tulemuse, kui tulemuses kasutada vahekaristust (ingl *gap penalty*). Võrdväärne oleks ühes jadas puudu olev element vastusesse lisada, bioloogia mõistes oleks see sama, kui osa geenist eemaldatakse või talle lisatakse osa juurde ehk toimub mutatsioon. Soodsaim joondus on selline, millele on antud hulk reegleid ja parameetreid, kuid pole olemas ainuõiget joondust, kuna see oleneb eeldustest, mille põhjal joondus tehakse. Vahe kasutatakse olukorras, kui on toimunud inertsioon või deletsioon. Vahed aitavad luua mudelile lähedasemaid joondusi, mis aitavad leida mustreid, millel on on otsija jaoks suurem tähendus. Vahe on oluline, kuna insertsioonid ja deletsioonid toimuvad tihti ühe mutatsiooni tegevusena. Vahekaristust kasutatakse, et karistada vahe kasutamist, olles osa joonduse skoorist, seega mõjutab vahekaristuse suuruse valimine lõpptulemusena saadud joondust. Suurema karistuse korral satuvad joondusesse vähem eelistatud elemendid, kuna proovitakse vältida vahede tegemist. Vahekaristus koosneb tavaliselt kahest osast: vahe alustamise karistus ning vahe pikendamise karistus. Viimane on tavaliselt väiksem, kuna on tõenäolisem, et juhtub näiteks üks inertsioon kümne nukleotiidiga kui kümme ühe nukleotiidi inertsiooni, sest viimast on raskem sooritada. Tavapärased vahekaristuse tüübid on: konstantne vahekaristus (ingl *linear gap penalty*, joonis 3) ja kaheosaline vahekaristus(ingl *affine gap penalty*, joonis 4). Konstantne vahekaristus on vahekaristus, kus iga vahe alustamise ja pikendamise karistused on omavahel võrdsed ja karistusena kasutatakse enamasti negatiivset arvu, mis tähendab, et kaastakse võimalikult väike arv vahesi. Kaheosalise vahekaristuse puhul antakse vahekaristusele ja vahe pikendamise karistusele erinevad väärtused: pikendamise karistus määratakse enamasti väiksem, kuna bioloogiliselt on tõenäolisem, et esineb üks suur vahe, kui samapaljude elementidega väikesed vahed. Võib täheldada, et lineaarne vahekaristus on kaheosalise vahekaristuse alamliik.

g = bk
kus g on vahe eest antav karistus;
b on iga vahe positsiooni eest antav karistus;
k on antud vahe pikkus (antud ahes olevate positsioonide arv);

Joonis 3. Konstantse vahekaristuse arvutamise valem [2].

g = a + bk
kus g on vahe eest antav karistus;
a on vahe avamise karistus;
b on iga vahe positsiooni eest antav karistus;
k on antud vahe pikkus (antud ahes olevate positsioonide arv);

Joonis 4. Kaheosalise vahekaristuse arvutamise valem [2].

g = a + b * log10(k) -c * log10(d)
kus g on vahe eest antav karistus;
a on vahe avamise karistus;
b on iga vahe positsiooni eest antav karistus;
c on evolutsioonilise kaugusega seotud kordaja;
k on antud vahe pikkus (antud vahes olevate positsioonide arv);
d on kahe järjestuse vaheline eeldatav evolutsiooniline kaugus PAM-ühikutes;

Joonis 5. Logaritmilise vahekaristuse arvutamise valem [2].

Järgnev lõik põhineb Remmi õpikul [2], kus tuuakse välja, et üks variant on kasutada ka logaritmilist vahekaristust (ingl *logarithmic gap penalty*, joonis 5), mis arvutatakse inertsioonide ja deletsioonide sageduse ja pikkuse tekke alusel, homoloogilises valgus. See on sarnane skoorimaatriksile, kus võetakse arvesse homoloogsete valkude joondusest tekkinud tegelike aminohapete asendus sagedus. Logaritmilise vahekaristuse puhul muudab evolutsioonilise kauguse suurenemine vahekaristuse väiksemaks. Evolutsiooniline kaugus on usaldusväärne, kuna eeldatakse, et evolutsiooniliselt lähestes järjestustes on juhuslikud vahed mittesoodustatud.

3.3.2 Täielik joondamine

Täielik joondamine (ingl *global alignment*, GA) eeldab, et kaks jada on peaaegu samad kogu järjestuse ulatuses, GA proovib kahte jada sobitada algusest lõpuni [6]. GA-d on lihtne implementeerida, kuid tema kasutamine on kasulik vaid siis, võrreldavate jadade vahel pole palju kustutamise ning lisamise protsesse [8]. Bioinformaatikas kasutatakse GA jaoks Needleman-Wunschi algoritmi, mis oli ühtlasi ka esimene DP rakendamine bioloogiliste jadade võrdlemiseks.

3.3.3 Kohalik joondamine

Kohalik joondamine (ingl *local alignment*, LA) otsib kahte osa, mis oleks üksteisele võimalikult sarnased ning ei keskendu üleüldisele sarnasusele [6]. Bioinformaatikas kasutatakse LA jaoks Smith-Watermani algoritmi, mis on Needleman-Wunschi edasi

arendus. LA peaks kasutama siis, kui eeldatakse, et võrdluse all olevad jadad sisaldavad ühte või mitut üksteisest eraldatud sarnast ala [13]. LA puhul kasutatakse tavaliselt positiivset punktiskeemi, st, et vahed ning mittevasted on negatiivse tulemusega ja vasted positiivsega. Algoritm erineb GA-st selle poolest, et teisenduskaugust arvutades lisatakse valikute hulka ka arv null. See tagab selle, et mittekonservatiivsed piirkonnad ei vii skoori üleliia palju alla.

3.3.4 Ülekattega joondamine

Ülekattega joondamine (ingl *semiglobal alignment, overlap alignment*, SA) on GA alamliik, kus on vahekaristused lubatud jada alguses ja lõpus. SA tuleks kasutada, kui eeldatakse, et võrreldavad jadad on sarnased kogu kokkulangeva ala osas [13]. SA peab hõlmama ühte järjestust täielikult või mõlema järjestuse otsa, seega paiknevad SA algus ja lõpp joondusmaatriksi servas [2]. SA saab hinnata mitmel viisil, üks variant on lisada karistus kõigile erinevustele jadade vahel, kuid kasutatakse ka varianti, kus alguse ja lõpu vahe karistus on 0 ehk punkte maha ei võeta [13]. SA on kasulik, kuna enamasti ei leita sarnasusi bioloogilise jadade täies ulatuses [13]. SA-d saab implementeerida kasutades GA-d, kui lisada juurde tingimus, et alguses ja lõpus olevaid vigu ei võeta joondamisel arvesse.

3.3.5 Järkjärguline joondamine

Järkjärguline joondamine (ingl *progressive alignment*, PA) on heuristiline meetod, mis leiab hierarhiliselt PWA hulgast kõige populaarseima [6]. PA puhul ei eemaldata joondamise hindamist optimiseerimis algoritmist, tema eeliseks on kiirus ja tõhusus [6]. PA-s võrreldakse järjestusi omavahel paariti ning klasterdatakse sarnasuse järgi, klaster (ingl *cluster*) näeb välja nagu fülogeneesipuu (ingl *phylogenetic tree*) ning klasterdamist nimetatakse dendogrammiks [7]. PA joondamise algoritmid on MSA leidmiseks kiiremad, kui vastavad DP algoritmid [2].

Järgnev lõik põhineb Remmi õpikul [2], kus kirjutatakse, et PA-s ei leita vastust kohe, vaid mitmed joonduse paarid joondatakse üksteisega järk-järgult. Selleks, et PA-s vähendada joonduses tekkivate vigade tõenäosust, kasutatakse fülogeneesipuud, mis on andmestruktuur kirjeldamaks bioloogiliste järjestuste sugulust võrgustiku kujul. Evolutsiooniliselt lähemate järjestuste joondamisel tekib vigu vähem, kui evolutsiooniliselt kaugemate joondamisel. Kuna PA alguses tehtud vead, näiteks vahe valesse kohta sisestamised jäävad joondusesse alles, sest vigu ei parandata, vaid lisatavaid joondusi üritatakse kohandada olemasolevatega, on oluline alguses joondada omavahel kõige sarnasemad jadad ning lõpus kõige erinevad jadad– just selle tagab fülogeneesipuu. Fülogeneesipuu rakendamine tähendab, et joondusi lisatakse puu harude hargnemise järjekorras. PA eesmärk ei ole saavutada optimaalset tulemust, seetõttu ei kasutata ka numbrilist skoori: kõik on protseduuri põhine, kui algoritm on töö lõpetanud, väljastatakse tulemus. Tulemuse

kvaliteet sõltub sellest, kui hästi on arvesse võetud evolutsioonist tingitud eripärasi, seega sõltub algoritmi tulemus paljudest parameetritest.

3.4 Naiivne meetod

Käesolevas töös mõeldakse naiivse meetodi all meetodit mingi kindla probleemi lahendamiseks, mis proovib läbi kõik võimalikud juhud. Näiteks, kui on vaja leida kahe jada vahel joondus, oleks üks variant läbi proovida koostada kõik võimalikud joondused nende vahel ja sealt valida parim. Oluliselt kasulikum oleks selle protsessi juures aga ära kasutada juba tehtud tööd, sellest täpsemalt järgmises peatükis (3.5).

3.5 Dünaamiline programmeerimine

DP on matemaatilise optimeerimise (ingl *mathematical optimization*) ja programmeerimise meetod, mille leiutajaks loetakse Richard Bellmani. Meetodi idee on jaotada suurem probleem väiksemateks alamprobleemideks, mille lahendusi hiljem omavahel kokku liites leitakse vastus probleemile, mille naiivne lahendamine oleks (ajaliselt) kulukam või võimatu. DP on sarnane jaga-ja-valitse algoritmidele, mõlemad jagavad probleemi väiksemateks osadeks. Erinevus kahe vahel seisneb selles, et jaga-ja-valitse käsitleb alamprobleeme eraldi, kuid DP käsitleb kõike koos. See tähendab, et DP alamprobleemide lahenduskäik on teineteisele sarnane, tänu sellele saab juba arvutatud tulemusi taaskasutada [14].

DP on peamiselt tavapärase rekursiooni optimeerimine kõikjal, kus kasutatakse rekursiivseid väljakutseid, millel on korduvad väljakutsed sama sisendi jaoks, saadakse seda optimeerida DP-ga [15]. Oluline on rekursiooni läbimisel alles hoida eelmistel läbimistel saadud tulemus ehk alam probleemi väärtus [16]. Näiteks, kui lahendatakse Fibonacci numbrite probleemi rekursiooniga, siis on selle ajaline keerukus eksponentsiaalne, kuid juba leitud vastuseid mällu jättes, muutub keerukus lineaarseks [15].

DP lahendused jaotuvad kahte liiki: alt-üles ja ülevalt-alla, nende peamine vahe seisneb selles, kuidas alamlahendusi salvestatakse [14]. Jätkates Fibonacci arvude näitega, saab DP-ga probleemile lahenduse leida kahel viisil: olgu funktsioonile antud parameeter n, siis alt-üles läbimise puhul hakatakse salvestama kasvavalt n-i suhtes ning ülevalt-alla läbimise puhul kahanevalt n-i suhtes. Ühtlasi võib öelda, et ülevalt-all on rekursiivne ning iga alam ülesannet läbitakse üks kord, kasutades mällu jäetud vahetulemust ülesande ja alam ülesannete lahendamiseks [16]. Alt-üles lahenduse puhul järjestatakse alamülesanded, väiksemast suuremani nii, et “üleval” jõutakse lahenduseni [16].

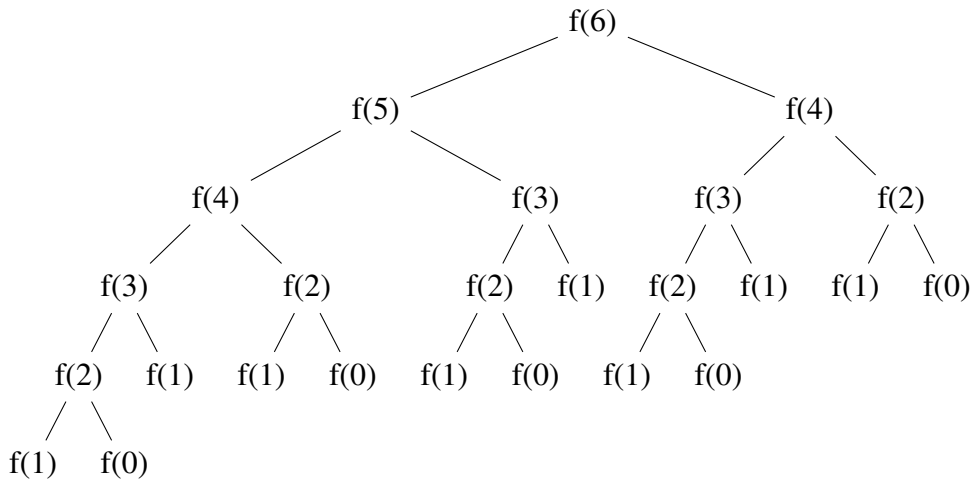
3.5.1 Kasutamise näited Fibonacci arvude arvutamiseks

Antud alampeatükis näidatakse DP kasutamise efektiivsust jooniste 6-8 abil, lisas 1 on joonistel 12-14 samas järjekorras välja toodud Pythonis kirjutatud näidislahendused. Näidete koostamisel on kasutatud funktsiooni f, mis võtab argumendiks nulli või positiivse täisarvu ning tagastab arvu, mis asub Fibonacci jadas argumendiga täpsustatud positsioonil. Arvutamisel kasutatavad seaduspärasused on väljatoodud valemites (1-3).

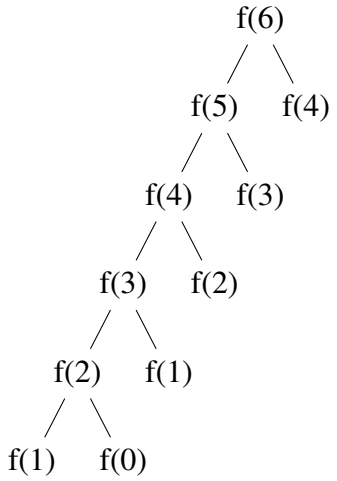
$$f(0) = 0. \tag{1}$$

$$f(1) = 1. \tag{2}$$

$$f(n) = f(n-1) + f(n-2). \tag{3}$$



Joonis 6. Fibonacci numbri leidmine asukohal n, kasutades naiivset lahendust.



Joonis 7. Fibonacci numbri leidmine asukohal n, kasutades DP ülevalt-alla lahendust.

indeks	0	1	2	3	4	5	6
väärtus	0	1	1	2	3	5	8

Joonis 8. Fibonacci numbri leidmine asukohal n, kasutades DP alt-üles lahendust.

Joonisel 6 on välja toodud kõigi sammude arv, mis tuleb teha, kui lahendust otsitakse naiivse funktsiooniga, tema rakendamine Pythonis on väljatoodud joonisel 12. Joonisel 7 on näha, kuidas DP-t kasutades sama arvutust ei tehta rohkem kui korra. Näiteks, kui f(4) on juba arvutatud, pole f(6) arvutamisel seda teist korda vaja arvutada, vaid vastus laetakse mälust. Antud lahendus kasutab ülevalt-alla lahenduskäiku, joonisel 8 on iseloomustatud ka DP alt-üles lahenduskäiku. Mõlema näite lahendused on Pythonis välja toodud vastavalt joonistel 13 ning 14. Viimase kahe vahel on näha, et esimene kasutab rekursiivset arvutusviisi, aga teine tabulaarset arvutusviisi.

3.6 Heuristilised meetodid

Tänapäevaks on kokku kogutud palju andmeid, näiteks Remm [2] on märkinud, et bioloogiliste järjestuste hulk andmebaasides sisaldab miljoneid järjestusi. Samuti on kogutud ka märkimisväärne hulk terviseandmeid. OHDSI (Observational Health Data Sciences and Informatics) kommuun on kokku kogunud 1.2 miljardit tervise kirjet[6]. DP algoritmid annavad täpse lahenduse ning kiiremini kui naiivsed meetodid, kuid siiski tekib olukordi, kus kiirus on olulisem kui saja protsendiline täpsus. Siinkohal kasutataksegi heuristilist algoritmi (HA), mis leiab täpse lahendus asemel ligikaudse lahenduse, et hoida kokku arvutamiseks kuluvat aega. HA rakendamise tüüpiliseks olukorraks on rändava müügimehe probleem (ingl *traveling salesman problem*) [2]. Antud töös HA-d ei rakendata, kuna eesmärk on leida parim viis haigustrajektooride võrdlemiseks.

Ghosh ja teised [6] on kirjutanud, et HA hülgab tulemuse korrektsuse ja tundlikkuse selleks, et saavutada kiireim lahendus. Teoses välja toodud proteiini andmebaasis on 100 miljonit jääki, mis 1000 elemendilise pikkusega moodustaks ligikaudu $10^1 1$ suuruse maatriksi. Sellise andmehulga töötlemine võtaks aega 10^4 sekundit ehk umbes kolm tundi. Kui otsingut teostada mitmete jadega, muutub aeg kiiresti oluliseks ressursiks. BLAST ja FASTA kasutavad HA-d, et üldsustada optimaalne kohalik sarnasus, mida kaks jada omavahel jagavad. Nende meetodite eesmärk on otsida võimalikult väike osa välju DP maatriksis, sisaldades kõrgete tulemustega maatrikseid. Sellist lähenemist tuntakse k-ennik (ingl *k-tuple*) meetodina. Meetod joondab kaks jada väga kiiresti, otsides esmalt identseid lühikesi joonduse alasi, ühendades need joondusesse läbi DP meetodite.

3.7 Algoritmid bioloogiliste jadade võrdlemiseks

Töö metoodika koosnob peamiselt teadmistest ning näidetest bioinformaatika valdkonnast, sest seal leiduvad andmed sarnanevad meditsiini andmetele. Üks oluline ühine omadus on see, et mõlemad on seriaalsed ehk kindlas järjekorras esinevad andmed, seega on võimalik nende peal rakendada jada joondamise (sõne võrdlus) algoritme, ühtlasi ka DP-t. Samuti sisaldavad mõlemad muudatusi (inertsioon, deletsioon), mis ei järgi kindlat reeglit, näiteks geeni variatsioonid või diagnoosi mitte esinemine. Oluline on märkida, et väljatoodud omadused ei pruugi kehtida kõigi bioinformaatikas käsitlevate andme tüüpide kohta, küll aga nende kohta, mida siin töös käsitletakse.

Kuna bioloogiliste andmete ülesmärkimiseks kasutatakse järjestust, võib iga erinevat kogumikku käsitleda kui ühte pikka sõne. Sellest tulenevalt käsitletakse töös sõnu jada ja sõne võrdväärsetena. Tänu sellele omadusele on bioinformaatika lahenduste loomisel võetud eeskuju sõna ning järjestike analüüsimise algoritmidest. Ka meditsiiniandmeid saab esitada jadana. Kuna terviseandmed ja bioloogilised andmed on sarnase ülesehitusega ning omadustega, on nad ka antud töös olulisel kohal. Siin peatükis tutvustatakse

[6] https://www.ema.europa.eu/en/events/common-data-model-europe-why-which-how

erinevaid viise sõnede omavaheliseks võrdlemiseks, alustades algelistest, lõpetades nendega, mis leiavad bioinformaatikas laialdast kasutust. Peatüki eesmärk on välja tuua, mis omadused on analüüsil olulised bioloogiliste andmete puhul ja läbi selle ka terviseandmete jaoks.

3.7.1 Kirjeldus

Bioinformaatikas on kõige olulisemal kohal bioloogiliste järjestuste uurimine, kus olulisemad neist on DNA, RNA ning valgud, mida märgitakse üles tähtega, sarnanedes inimtekstile [2]. Eelnimetatud järjestusi nimetatakse ka makromolekulideks ning nende analüüsi meetodeid analüüsides proovib autor leida kõige optimaalsemat viisi meditsiini andmete analüüsimiseks. Bioinformaatika meetodite arendamisel võeti alguses aluseks sõnevõrdlus algoritmid, kuid see polnud piisav, sest need ei võtnud arvesse andmete bioloogilist sisu. Sellepärast hakati looma algoritme spetsiaalselt bioloogiliste andmete jaoks. Esimene neist oli Needleman-Wunschi algoritm (NWA), mis kasutas juba olemasolevaid teadmisi informaatikas: muutmiskaugus ning DP, lisades juurde ka meetodid bioloogilise sisuga arvestamiseks. DNA sekveneerimine on protsess, mille käigus saadakse **väga väike osa** kogu DNA ahelast. Üks bioinformaatika valdkondi lahendabki probleemi, mis tegeleb terve DNA kokku panekuga sekveneerimiselt saadud andmetest.

3.7.1.1 DNA

DNA on eukarüootsete organismide rakutuumas leiduv aine, mis määrab, milliseid valke raku sees toodetakse. Meioosi käigus kopeeritakse DNA mõlemasse rakku ning meioosi käigus antakse mõlemalt vanemalt edasi pool nende DNA-st, seega on DNA ühtlasi ka pärilikkust edasi kandev aine ning pakub huvi mitmele bioloogia valdkonnale. Kuna DNA alusel on võimalik leida otsitav RNA (matkides transkriptsiooni) ning RNA kaudu (matkides translatsiooni) ka vastavad valgud, puudub vajadus RNA ja valkude eraldi sekveneerimiseks, seetõttu on algandmetena kõige rohkem DNA sekveneerimise tulemusi.

DNA eksisteerib enamus ajast biheeliksina ehk tal on kaks haru, sellegi poolest märgitakse järjestusena üles ainult üks haru, mille teeb võimalikuks DNA-le omane pöördkomplementaarsus (ingl *reverse complementarity*). Pöördkomplementaarsus tähendab, et vastas olevas harus A-le vastab alati T ja vastupidi ning vastas olevas harus G-le vastab alati C ja vastupidi. DNA ahelad on suunaga ahelad, mida märgitakse vastavalt desoksüriboosi molekuli otsas vabaks jäävale süsinikuaatomile kas 5’ ots või 3’ ots ning oluline on, et ülesmärkimisel alustatakse alati 5’ otsast [2]. Pöördkomplementaarsus saavutatakse, kui üks haru poole keeratakse teist pidi ning kõik lämmastikualused asendatakse nende vastanditega. Seega, võttes sama DNA 3’ haru ning rakendades sellele pöördkomplementaaruse, on ta täpselt ühesugune antud DNA 5’ haruga.

DNA järjestusi märgitakse üles kasutades nelja sümbolit A, C, G ja T, mis vastavalt tähistavad DNA nukleotiidides leiduvaid lämmastikaluseid: adenosiin, tsütidiin, guanosiin ja tümidiin. Vahest kasutatakse ka tähist N märkimaks, et antud asukohal võib esineda ükskõik millne neljast eelnevalt mainitud sümbolist. DNA motiivide kirja panemiseks kasutatakse Rahvusvahelise Puhta ja Rakenduskeemia Liidu (IUPAC) sümboleid, mida on kokku viisteist [2].

Järgnev lõik põhineb Langmead jt kursusele [17], kus räägitakse, et DNA sekveneerimiselt saadud jadade kokkupanemisel on kaks erinevat viisi, lihtsaim neist on genoomi kokku panemine (sama liigi) viitegenoomi alusel. Selle jaoks on vaja leida jada asukoht viitegenoomist lähtuvalt ning nõnda juppe kokku pannes saadaksegi sama liigi uue isendi genoom. Viitegenoomi puudumisel on kokku panemine oluliselt raskem, sest puudub nii öelda kaart, millele lähtuvalt saab iga juppi koheselt omale kohale panna. Ilma viitegenoomita kokkupanemine ehk *de novo* kokkupanek toimub sarnaseid jadasi omavahel ühendades, see protsess on väga aeglane, sest liikuda tuleb tüki haaval. Tasub mainida, et selline kokkupaneku viis ei tule hästi toime korduvate osadega genoomis, sest raske on kindlaks määrata, kas mingi osa kordub näiteks kaks või kolm korda. See mõjutab ka viitegenoomi kaudu joondamist, sest läbi *de novo* on viitegenoom koostatud.

3.7.2 Täpne sobitamine

Täpse mustri sobitamise algoritm (ingl *exact pattern matching*) kahe sõne sobitamiseks koosneb lähteallikast, kust otsitakse ning mustrist, mida otsitakse. Täpne siin kontekstis tähendab seda, et otsitakse just seda mustrit, mis on ette antud. Kõige lihtsam implementatsioon on niinimetatud naiivne algoritm, kus proovitakse kõik mustri võimalikud sobivused lähteallika vastu läbi. Selline lahendus aga on ajaliselt kulukas. Teised implementatsioonid, nagu näiteks kmeer indeks, Boyer-Moore aloritm ning Knuth-Morris-Pratt algoritm, võtavad arvesse juba tehtud tööd ning oskavad selle arvelt vahele jätta võrdlusi, jõudes kiiremini soovitud lahenduseni. Tehtud töö all mõeldakse lähteallika eeltöötlemist (ingl *pre-processing*), kmeer indeksi puhul salvestatakse kõik võimalikud m pikkused jadad ning nende asukohad lähteallikas. Samuti kasutab Boyer-Moore algoritm ära joonduse läbimisel kogutud teavet, seda halva tähe reegli (ingl *bad character rule*) abil[7]. Täpse mustri sobitamise algoritme kasutatakse bioinformaatikas vähe, sest bioloogilised andmed võivad sisaldada lünki ning vigu ja see muudab nimetatud algoritmide efektiivsuse madalaks.

Järgnev lõik põhineb Langmeadi jt kursusel [17], kus räägitakse, et täpset sobitamist kasutatakse bioinformaatikas genoomi koostamisel viitegenoomi abil. Näiteks soovitakse joondada suvalise inimese genoom, siis selle jaoks kasutatakse lähteallikaks inimese

[7]Boyer-Moore. Ben Langmead `http://www.cs.jhu.edu/~langmea/resources/lecture_notes/boyer_moore.pdf` (25.11.2020)

viitegenoomi. Selle jaoks on viitegenoom varasemalt ära indekseeritud, kasutades kmeer indeksit ning selle abil on võimalik üles leida sarnasuspiirkonnad ja nende peal rakendades ligikaudset sobitamist on võimalik kindlaks teha, kas tegu on õige asukohaga genoomis. Oluline on märkida, et juba indekseeritud lähteallikat on võimalik taaskasutada ehk eelneva näite jätkuks, saab koostatuga joondada kõigi uute inimeste genoomi.

3.7.3 Ligikaudne sobitamine

Palju kasulikumaks on bioloogiliste andmete puhul osutunud ligikaudsed mustri sobitamise algoritmid (ingl *approximate pattern matching*), kuna nende jaoks pole oluline üksühene vastavus lähteallikaga vaid võimalikult lähedane üleüldine sarnasus. Selline lähenemine on sarnasem evolutsiooni protsessile. Kuna kaal langeb kogu sekventsi sarnasusele, ei oma üksikud vead nii suurt kaalu. Ligikaudsed mustri sobitamise algoritme kasutatakse ka andmebaaside otsinguteks ja plagiaadi tuvastamiseks. Levinum viis ligikaudse mustri sobitamise algoritmi rakendamiseks on kasutada teisenduskaugust, mis väljendab kahe jada omvahelist erisust.

Bioinformaatikas kasutatakse teisenduskauguse arvutamiseks vahekaristust ning skoorimaatriksit (vaata lähemalt peatükk 3.7.4), et võtta arvesse bioloogilist sisu. Lisaks kasutatakse ka erinevaid joondamise liike vastavalt joondatavate andmete eripärale ning soovitud tulemusele. DP-t kasutatakse selle jaoks, et ei peaks koostama kõiki võimalikke joondusi, kui soovitaks võrrelda kahte jada, vaid selle asemel leitakse parim teekond ning tagastatakse selle abil optimaalseim tulemus.

3.7.3.1 Paarisjoondamine kasutades dünaamilist programmeerimist

Esimesed arvuti põhised meetodid bioinformaatilise infromatsiooni omavaheliseks võrdlemiseks olid DP algoritmid. Sellepärast, et andmete hulk on küllaltki suur, tuleb analüüside jooksutamisel arvestada märkimisväärse aja kuluga. Naiivne meetod kahe järjestuse n ja m joondamiseks oleks kõik võrreldavad alamjadad välja arvutada ning leida nende hulgast sobivam ehk optimaalseim lahendus. Taolise lahenduse ajaline keerukus oleks aga $O(2mn)$, sest samme on kokku m korda n ning igal sammul saame valida elemendi kas n-ist või m-ist, seega kaks sammu. Esimese rahuldava ajalise keerukusega algoritmi pakkusid 1970. aastal välja Saul B. Needleman ja Christian D. Wunsch, mida tuntakse kui NWA [18]. NWA ajaliseks keerukuseks on O(nm), kuna kõik sammud on vaja läbi käia (n korda m) ja peale seda liigutakse viiteid pidi tagasi algusesse. Kuna enamasti on vaja leida konserveerunud piirkonda valgust, tegid Temple F. Smith ja Michael S. Waterman NWA algoritmi edasiarenduse, mida tuntakse kui Smith-Watermani algoritm (SWA) [19]. Peamine erinevus NWA ja SWA vahel on välja toodud joonisel 9. Erinevus seisneb selles, et SWA puhul iga välja skoor ei saa jääda alla 0-i, seega ei karistata vahepeal leiduvaid mittevasteid nii rangelt, kui seda teeb NWA. Järgnev lõik tugineb Ghosh jt õpikul [6], kes on kirjutanud, et DP algoritmid on kesksel kohal analüüsimaks jadasi arvutitega.

(NWA) Täielik joondus	(SWA) Kohalik joondus
F(i, j) = max(F(i-1, j-1) + s(xi, yj), F(i-1, j) - d), F(i, j - 1) - d),)	F(i, j) = max(F(i-1, j-1) + s(xi, yj), F(i-1, j) - d), F(i, j - 1) - d), 0)

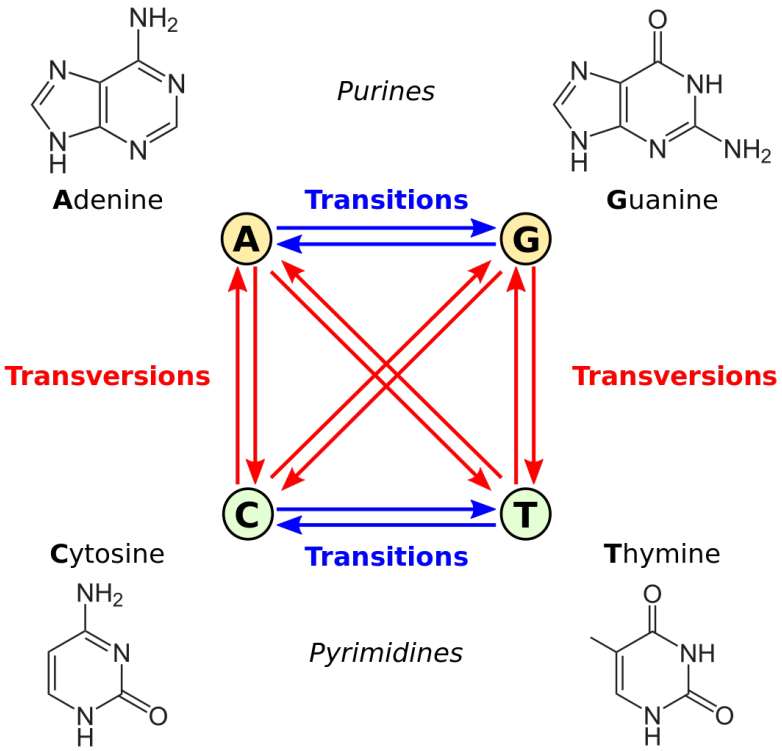
Joonis 9. Joondusmaatriksi sammu väärtustamine täieliku ja kohaliku joonduse algoritmi jaoks [6].

Needleman ja Wunsch lõid 1970. aastal esimese DP algoritmi, mille eesmärk oli võrrelda kahte jada. See on baas algoritm, mida kasutavad enamus paaris joonduse tööristu. Kuna jada joondus mängib bioloogias mitut rolli, on selle populaarus olnud vahelduv. DP on efektiivne, rekursiivne meetod analüüsimaks läbi kõik võimalikud lahendid ning leidmaks sealt optimaalse skoori. DP koosneb tavapäraselt kolmest osast: rekursiivne relatsioon, tabelarvutus ning joonduse leidmine (ingl *traceback*). Iga automaatse meetodi töötamiseks on vaja täpselt seletada, millised eeldused sellele on seatud. Vaja on määrata kindel vahekaristus, valida skoorimaatriks, sellist kombinatsiooni nimetatakse tavaliselt skoorimise skeemiks (ingl *scoring scheme*). Eesmärk on joondada kaks jada (x1, x2, ... xn ja y1, y2, ... yn) ja selle põhjal koostada maatriks F = m n, kus F(i, j) on GA optimaalse teekonna skoor[alamjadade x1, x2, ... xn ja y1, y2, ... yn vahel. LA puhul on skooriks kõrgeima väärtusega väli. Joonisel 9 on välja toodud DP algoritmi rekursioon nii GA kui ka LA jaoks, mis kasutab konstantset vahekaristust (d) ning mille baassammudeks on F(0,0) = 0, F(i,0) = -id ja F(0,j) = -jd. Veeru väärtuse arvutamiseks kasutatakse funktsiooni s, mida enamasti kasutatakse skoorimaatriksist asenduse väärtuse leidmiseks (vaata peatükk 3.7.4). Tabelarvutustes (ingl *tabular computation*) alustatakse punktist (0, 0) ning arvutatakse rea kaupa, igas punktis (i, j) jäetakse meelde viide eelnevale suurima skooriga väljale. GA puhul on lõpppunktiks alati (0,0), kuid kohalik joondus lõpeb siis, kui käsitletavas veerus puudub viide järgnevasse veergu.

3.7.4 Skoorimaatriks

Järgnev lõik on refereeritud Ghosh jt õpikust [6], kes on kirjutanud, et joondades kahte jada, proovitakse selgitada, kas nende vaheline joondus on tekkinud juhuslikult või tegu on sarnaste jadadega. Selle jaoks on igale võimalikule muudatuste paarile võimalik määrata kindel väärtus ehk skoor. Kõige lihtsam on seda saavutada, määrates skoori vastete ja mittevastete paaridele, taolist maatriksi kutsutatakse ühtseks maatriksiks (ingl *unitary matrix*). Nukleotiidide joondamiseks võib ühtne maatriks olla piisav, kuid mitte aminohapete jaoks, kuna muudatused aminohapetes sisaldavad tavapäraselt rohkem teavaet, kui muudatused nukleotiidides.

Üks võimalus DNA elementide muutumise ühtse maatriksi koostamiseks oleks iga nukleotiidi vaste eest anda punkt ning iga mittevaste ja vahe eest punkt maha võtta. Selline lahendus aga ei arvesta bioloogilise sisuga. DNA mutatsioone on kahte tüüpi: **transitsioon**, mis leiab aset puriinide vahel või pürimiidide vahel ning **transversioon**, mis toimub puriini ja pürimiidi vahel [2]. Joonisel 10 tuuakse detailselt välja kõik mutatsioonid ja nende suhted. Kuigi transversioon mutatsioone on kaks korda rohkem kui transitsioone, leidub kahe suvalise inimese genoomi võrreldes kaks korda rohkem transitsioone kui transversioone [17]. See on tingitud vastavate DNA molekulide struktuurist, sest transversiooni käigus muudab aine struktuuri (ühelt rõngalt kahele või vastupidi), seega on transitsiooni lihtsam teha ja seepärast leidub teda oluliselt rohkem. Eelnevat arvesse võttes on võimalik koostada maatriksi (joonis 11), mis võtab seda arvesse ning on seetõttu bioloogilise sisu mõttes täpsem.



Joonis 10. DNA mutatsioonid[8]

.

Aminohapete puhul tuleb arvesse võtta, et valkude joondamisel ei ole tegu võrdväärsete tähtedega, kuna igal aminohappel on erinevad füüsikalised ja keemilised omadused, mis seavad antud valgu asenduskiiruse [2]. See ongi põhjuseks, miks ühtne maatriks valkude

[8]Transversion. Wikipedia. https://en.wikipedia.org/wiki/Transversion (17.07.2021)

	A	C	G	T	-
A	0	4	2	4	8
C	4	0	4	2	8
G	2	4	0	4	8
T	4	2	4	0	8
-	8	8	8	8	

Joonis 11. Ühtne maatriks nukleotiidide jada (täielikuks) joondamiseks, kus "tähistab vahe [17].

joondamisel ei ole piisav. Remm [2] on välja toonud, et valkude skoorimaatrikseid on proovitud erinevaid liike, kuid tänapäeval kasutatakse vaid aminohapete asendussagedust arvestavaid maatrikseid. Ta mainib veel, et selline maatriks saavutatakse jälgides homoloogsetes valkudes aminohapete asenduskiirust, kuna aminohapped, mis homoloogses valgus kergesti asenduvad, on sarnased. Esimene, kes taolise idee realiseeris, oli Margaret Dayhoff. Lisaks tuuakse välja, et sellisel põhimõttel koostatud maatriks on põhimõtteliselt valkude evolutsiooni matemaatiline mudel ning taolisi maatrikseid nimetatakse log-odds-tüüpi skoorimaatriksiteks (ingl *log-odds scoring matrix*). Kaks populaarsemat sedatüüpi maatriksit, PAM ja BLOSUM on väljatoodud vastavalt peatükkides 3.7.4.1 ja 3.7.4.2. Log-odds-tüüpi skoorimaatriksi nimi võib olla eksitav, sest tavapäraselt mõeldakse selle all šanssi, mis iseloomustab suhet soodsate ja ebasootsate võimaluste vahel. Hetke kontekstis tähistab ta lihtsalt kahe tõenäosuse vahelist suhet, kuid aminohapete võrdlemisel tähistab see kahe aminohappe omavahelist asendumise sagedust evolutsiooni käigus [2].

3.7.4.1 PAM

Järgnev lõik tugineb Remmi õpikul, [2], kes on välja toonud, et PAM-tüüpi skoorimaatriks (ingl *point accepted mutations matrix*) on skoorimaatriks, milles tähtede esinemissagedused arvutatakse teoreetiliselt. Skoorimaatriksi nimes olev arv näitab, kui suure evolutsioonilise kaugusega järjestusi kirjeldatakse. Iga maatriks on seotud kindla evolutsioonilise ajaperioodiga ja selleks, et võrrelda erinevate evolutsiooniliste kaugustega valke, on koostatud erinevad skoorimaatriksid. Ajaperioodi võiks mõõta aastates, kuid kuna erineva funktsiooniga valgud evolutsioneeruvad erineva kiirusega, on kasulikum mõõtmisi sooritada aminohapete omavaheliste asenduste protsendina. PAM-tüüpi skoorimaatriksis on taoliseks ühikuks PAM (ingl *point accepted mutations*), mis näitab, mitu aminohappe asendust on toimunud uuritava valgu järjestuses 100 aminohappe kohta. PAM-tüüpi skoorimaatrikseid on olemas vahemikus 1-500, kus PAM1 tähendaks järjestust, kus 1% tähtedest on asendatud, seega on ta valikust vähima evolutsioonilise kaugusega. Maatriksi koostamiseks leitakse alguses asenduste tõenäosus ühe evolutsioonilise kauguse ühiku jaoks ehk PAM1. Ülejäänud variandid leitakse PAM1 asetustest matemaatiliselt, korrutades asenduste arvu soovitud vahemikuga. Näiteks PAM250 maatriksi korral korrutatakse PAM1 asetused arvuga 250 ning pärast teisendatakse log-odds kujule.

3.7.4.2 BLOSUM

BLOSUM (ingl *Block substitution matrix*) skoorimaatriksi arvutamisel jäetakse välja ekstrapoleerimise probleem ning selle asemel loendatatakse asendamissagedusi otseselt, tervete joonduste asemel kasutatakse sarnaseid või identseid alamjadasid, mis ei sisalda vahesi [20]. BLOSUM loendab joondusest valkude otsesed asetused, kasutades PAM-tüüpi skoorimaatriksile vastupidist nummerdamis süsteemi, kus number BLOSUM maatriksi nime järel näitab, maatriksi koostamiseks kasutatavate valkude (mille sagedus kokku loeti) identsust [2]. Üks levinumaid variatsioone on BLOSUM62 maatriks, see tähendab, et ta põhineb joondustel, mis on vähemalt 62% sarnased. Selle maatriksi põhifookus on võimalikult hästi mõõta kahe proteiini vahelisi erinevusi, eriti kahe kaugelt suguluses oleva valgu vahel [8].

Järgnev lõik põhineb Ghosh jt [6] õpikul, kelle sõnul on BLOSUM maatriks vastandiks PAM maatriksi ideele, et evolutsioonilised määrajad on ühtsed üle terve proteiini järjestuse. See aga ei vasta tõele, kuna evolutsioonilised määrajad on madalamad sarnastes regioonides ning kõrgemad mitte sarnastes regioonides. BLOSUM-i koostamiseks kasutati BLOCK andmebaasi, et väga sarnastest jadade regioonidest leida erinevusi. Maatriksi koostamiseks võeti kõik järjestused BLOCK andmebaasist ning igale järjestusele kohta arvutati summa, kui tihti erinevad aminohappe paarid esinevad omavahel koos kindlates sarnastes regioonides. Sagedused tuleb üles märkida sagedustabelisse ning tõenäosus sarnasuseks arvutatakse selle põhjal sarnaselt log-odds maatriksile.

4 Trajektooride leidmine

Uurimustöö praktilise väljundina katsetati erinevaid joondamise meetodeid sarnaste haigustrajektooride leidmiseks. Selle jaoks analüüsiti kunstandmeid, mis pärinesid Valdase [4] koostatud generaatorist. Antud peatükis tuuakse välja parim kogum meetodeid leidmaks sarnaseid haigustrajektoore patsiendi diagnooside kogumikust ning parameetrid, mis on muutuvad, peatükis (5). Analüüsimise jaoks loodud koodile on viide lisas 2. Kood kasutab läbivalt Pythoni programmeerimise keelt (versioon 3.9.6.), lisaks on kasutuses ka üks väline teek: tqdm (versioon 4.62.0), ülejäänud kasutusolevad pärinevad Pythoni standard teegist ehk nad on seotud Pythoni enda versiooniga.

4.1 Ettevalmistus

Katsetamiseks genereeriti kunstandmete kogumik, tuhande (kunst) inimese diagnooside jadaga. Andmete genereerimiseks kasutati Valdase [4] lõputöös valminud juhuslike diagnooside trajektooride generaatorit. Kuna generaator andis trajektoorid RHK-10 formaadis, kasutati analüüsimisel seda formaati. Valdase [4] töö raames on koostatud liides, mis võimaldab neid andmed viia ka OMOP formaati. Kuna generaatori väljund on juba RHK-10 formaadis, siis kasutatakse seda ka analüüside tegemisel. Generaator omab lisafunktsioone andmete kuvamiseks, kuid neid siin töös ei kajastata.

Järgnevas lõigus kirjeldatakse generaatori kasutamist mis põhineb Valdase [4] lõputööl. Testandmete genereerimiseks kasutatakse generaatori juurkaustas olles, kasutada käsurea koodi `python main.py -p 1000`, mis loob projekti *output* kausta faili nimega *diagnoses.csv*. Kuna iga genereeritud kogumik on erinev (juhuslik), on katsetamisel kasutatud kogumik projektiga kaasas, et tagada tulemuste taastekitamise võimalus. Genereeritud andmed esitatakse tabelina, kus iga rida vastab ühele patsiendile. Iga patsiendi kohta on teada tema sugu, sünni- ja surmapäev (surmapäev vaid siis, kui isik on surnud), vanus ning diagnoosi trajektoorid, mis tuuakse välja neljas veerus: Chapter, Subchapter, Section ja Subsection ehk vastavalt eesti keeles: Peatükk, Alampeatükk, Jaotis ja Alamjaotis, vastavalt RHK-10 hierarhiale. Hierarhia on jaotatud nii, et samal indeksil asuv peatükk näitab diagnoosi esimese taseme kuuluvust ehk kuuluvust peatükki, alampeatükk kuuluvust alampeatükki jne. Märkimisväärne et kasutatud generaator algseadistuses ei tagastanud alamsektsioone.

4.2 Kasutatud metoodika

Jadade joondamiseks kasutatakse LA algoritmi, kust on välja jäetud tagasipöördumise samm, sest antud programmis on vaja teada, kui sarnased kaks jada teineteisele on. Karistusskeemina kasutatakse 1 punkt vaste ning 0 punkti vahe ja mittevaste eest, kuna haigustrajektooride puhul ei mängi asukoht nii suurt rolli, kui näiteks bioloogiliste

andmete puhul, sest kahe huvipakkuva diagnoosi vahel võib olla ka mittehuvipakkuvaid diagnoose ning nende kogus pole ette teada. Sellepärast ei soovita karistada vahepealseid lünki ning mittevasteid. Taoline algoritm on kohati sümbioos LA-st ning GA-st, sest huvi piirkonnaks võib osutuda kogu maatriks. Võrdlusel annab jada joondamine olulisuse tekkimisjärjekorrale ehk pole ainult oluline diagnooside esinemine, vaid ka järjekord. Joondamise algoritm asub analüüsi koodi repositooriumis *local_alignment.py* failis.

Ülejäänud osa koodist asub failis *main.ipynb*, kus laetakse algselt sisse patsientide diagnooside andmed (*diagnoses.csv*) ning nende loomisel kasutatud trajektoorid (*trajectories.py*). Diagnoosiandmetest luuakse *sections* massiiv, mis sisaldab kõiki patsiente trajektooridena, jaotise tasemel, sest alamjaotise andmeid antud seadistuses generaator ei tagastanud. Kunstandmete loomisel rakendab generaator trajektoore juhuslikult, seega kõiki neist ei pruugi olla rakendatud. Seejärel laetakse joondamise funktsioon ning käivitatakse funktsioon *get_trajectories*, mis tagastab leitud klastrid ja nendele vastavad trajektoorid.

Funktsiooni *get_trajectories* töökäik on jagatud osadeks, mõnda nendest on võimalik efektiivsuse saavutamiseks ka omavahel ühendada, kuid arusaadavuse mõttes on kõik paigutatud eraldi funktsioonidesse. Esmalt leitakse funktsiooni sees mitu korda mingi diagnoos kogu *sections* massiivi ulatuses esineb. Seejärel koostatakse massiiv, mis sisaldab seatud piiridesse jäävaid diagnoose. Piiri idee on välja lõigata statistiliselt ebaolulisi diagnoose st kui mõni diagnoos esineb liiga vähe, on ebatõenäoline, et ta osaleb mõnes trajektooris, samuti kui mõnda diagnoosi leidub liiga palju, ei ütle ta väiksema rühma kohta midagi. Oluline on, et alampiiri ei saa ka väga kõrgele tõsta, kuna tekkivad klastrid ei pruugi olla väga suured. Seejärel eemaldatakse kõigilt trajektooridelt diagnoosid, mis ei kuulu motiivi, et järgmisel sammul (joondamisel) oleks võimalikult vähe müra. Järgnevalt leitakse eelnvevalt mainitud LA algoritmi rakenduseks loodud funktsiooniga sarnasuse skoor kõikide võimalike patsientide paaride vahel. Klastrisse lisatakse kõik paarid, mille omavaheline joondamine annab tulemuseks viis või suurem. Peale klasterdamist jäetakse alles klastrid, millel on vähemalt kuus trajektoori. Viimase sammuna puhastatakse trajektoorid ning selle jaoks eemaldatakse diagnoosid, mis antud trajektoori klastris esinesid vähem kui kaks korda.

5 Tulemuste analüüs

Funktsiooni *get_trajectories* parameetritest muudeti motiivi loomisel kasutatavat sarnasuse skoori piiri. Tulemuse hindamiseks vaadeldi väljundina saadud iga klastri patsientide trajektoore omavahel, arvutades klastri sees oleva iga võimaliku paari sarnasuse skoori keskmine, kasutades eelmise peatükis mainitud LA funktsiooni. Vältimaks ebaolulisi trajektoore, hüljati võrdlusest need, mille pikkus oli alla kolme elemendi. Saadud tulemustest saadi omakorda keskmine, mille alusel võrreldi erinevate parameetrite tulemusi. Mida kõrgem on saavutatud skoor, seda parem parameetrite kogumik. Antud peatükis esitatud tulemused on kõik kümnendikeni ümardatud.

Parim leitud tulemus saavutati siis, kui motiivi piiriks seati $(82, 100)$ ehk mudelis kaasatud diagnoosil pidi olema vähemalt 82 vastet ning võis kokku olla kuni 100 (kaasa arvatud) vastet. Mainitud tulemuseks saadi 3.62. Hindamaks parimat tulemust, leiti kõigi võimalike trajektooride omavaheliste sarnasuste keskmine: 1.27. Seega suudab leitud mudel koondada omavahel patsientide trajektoorid, mis on üksteisele oluliselt sarnasemad, kui kaks suvalist patsienti.

Hindamaks generaatori poolt kasutatud signaali kinnipüüdmist, võrreldi eelnevalt mainitud LA funktsiooniga kõikide võimalike signaali trajektooride ning leitud patsientide trajektooride sarnasusi. Selle jaoks koguti kõik nimetatud paarid, mille sarnasus skoor oli kaks või rohkem. Tabelis 1 on tulemused välja toodud teises tulbas. Võrdluseks on lisatud $(2, 100)$ mudel, mis leiab signaalis palju vasteid, kuid see eest on ebatäpsem. Ebatäpsus on kohati tingitud suuremast trajektooride arvust.

Tabel 1. Funktsiooni *get_trajectories* tulemuste võrdlus.

	Sarnasus signaaliga (>=2 vastet)	**Paaride keskmine sarnasus**	**Trajektoore kokku**
Kõik paarid	9417	1.28	
(2, 100)	36	2.79	75
(82, 100)	0	3.62	34

Saavutatud mudel pole kindlasti parim signaali kinnipüüdmiseks, samuti tuleb arvestada, et kasutatud parameetrid ei pruugi kehtida mõne teise genereeritud andmehulga kohta. Lisaks on tähtis funktsioonile etteantud patsientide arv ning kui see on liiga suur, ei leiaks mudel soovitud tulemisi. Antud probleemi aga ei saa lahendada lihtsa kordajaga, sest andmehulgas, mis on n korda suurem, ei esine iga diagnoosi n korda rohkem.

Kokkuvõte

Töö eesmärk leida sarnaseid haigustrajektoore saavutati töös väljatoodud piirangute raames, käsitledes genereeritud patsientide diagnooside trajektoore. Küll aga ei leitud head meetodit avastamaks generaatori poolt lisatud signaale. Töö käigus tutvustati meditsiini andmeid ning nende juurde käivat keerukust. Toodi välja bioinformaatikas leiduvaid probleeme ning kuidas need seal valdkonnas on lahendatud. Selle põhjal tutvustati seriaalandmete võrdluse põhimõtteid, kasutades jada joondamist.

Töös tutvustatud meetodit saab edasi arendada lisades sinna rohkem meditsiini andmeid, kõige lihtsam seda saavutada on kasutada OMOP andmemudelit. Hetke analüüs arvestab ainult diagnooside ja nende tekkimis järjekorraga, kuid võimalik oleks ka sisse arvestada nende tekkimise aega. Võimalik on ka uurida diagnooside esinemissagedust suuremas kunstandmete kogumikus või pärisandmetes, et määrata nende esinemisele kindel kaal. Samuti saaks analüüsi rohkem konteksti lisada, kui sarnaselt eelmisele soovitusele analüüsida erinevate diagnooside koosesinemise sagedust ja sellele määrata kaal. Kindlasti oleks vaja funktsiooni parameetrid asendada relatiivse muutujaga, et leida soovitud tulemusi ka suurtest andmekogumikest.

Viidatud kirjandus

[1] U.S. National Library of Medicine. Basic Local Alignment Search Tool. https://blast.ncbi.nlm.nih.gov/Blast.cgi (25.02.2020).

[2] Remm M. Bioinformaatika. *Tartu Ülikooli Kirjastus*. 2015.

[3] Sarkar T. Synthetic data generation — a must-have skill for new data scientists. *Towards data science*. https://towardsdatascience.com/synthetic-data-generation-a-must-have-skill-for-newdata-scientists-915896c0c1ae (30.07.2021). 2018.

[4] Valdas A. Juhuslike diagnooside trajektooride generaator. *TARTU ÜLIKOOL, 2021, Arvutiteaduse instituut, Informaatika õppekava*. https://comserv.cs.ut.ee/home/files/valdas_l%C3%B5put%C3%B6%C3%B6.pdf?study=ATILoputoo&reference=9222B4B41883A489B5BF9C871F17B66AB20BAF38 (30.07.2021).

[5] Observational Health Data Sciences ja Informatics. The Book of OHDSI. 2019.

[6] Ghosh Z. ja Mallick B. Bioinformatics: Principles and Applications. *Oxford University Press*. 2020.

[7] Claverie J.M. ja Cedric Notredame N. Bioinformatics For Dummies 2nd Edition. *For Dummies*. 2006.

[8] Sadek A. H. Bioinformatics: Principles and Basic Internet Applications. *Trafford Publishing*. 2006.

[9] Lember J., Matzinger H. ja Vollmer A. Optimal alignments of longest common subsequences and the https://arxiv.org/pdf/1407.1233.pdf. 2014.

[10] Käärik R. ÜLDISTATUD TEISENDUSKAUGUSE ARVUTAMINE 2006. http://www.quretec.com/u/vilo/edu/Students/Reina_Kaarik/Semestrit88.pdf (10.05.2021).

[11] Kececi M. Bioinformatics Tools: Introduction to bioinformatics. *Kececi M.* 2019.

[12] Nadia Pisanti. Algorithms Foundations *Encyclopedia of Bioinformatics and Computational Biology Elsevier*. 2019, lk 1-4.

[13] Durand D. Computational Genomics and Molecular Biology. *Carnegie Mellon University 2015*. http://www.cs.cmu.edu/~durand/03-711/2015/Lectures/PW_sequence_alignment_2015.pdf (10.12.2020).

[14] Tutorialspoint. Dynamic Programming. https://www.tutorialspoint.com/data_structures_algorithms/dynamic_programming.htm (10.12.2020).

[15] Geeks For Geeks. Dynamic Programming. https://www.geeksforgeeks.org/dynamic-programming/ (10.12.2020).

[16] Fišel M. Dünaamiline Programmeerimine. https://panopto.ut.ee/Panopto/Pages/Viewer.aspx?id=6339c921-0ab2-4e02-8079-dc81e7c66789&query=d%C3%BCnaamiline%20programmeerimine (10.12.2020).

[17] Langmead B. ja Pritt J. Algorithms for DNA Sequencing by Johns Hopkins University. https://www.coursera.org/learn/dna-sequencing (28.07.2021).

[18] Needleman B. S. ja Wunsch D. C. A general method applicable to the search for similarities in the ar *Journal of Molecular Biology*. 1970, nr 48, lk 443-453.

[19] Smith F. T. ja Waterman S. M. Identification of common molecular subsequences. *Mournal of Molecular Biology*. 1981, nr 147, lk 195-197.

[20] Pearson R. W. Selecting the Right Similarity-Scoring Matrix 2013. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3848038/ (25.02.2020).

Lisad

I. Fibonacci arvude arvutamise implementatsioon Pythonis

```
def fibonacci_naive(index):
    if index == 0 or index == 1:
        return index
    return fibonacci_naive(index - 1) + fibonacci_naive(index - 2)
```

Joonis 12. Fibonacci numbrite arvutamise naiivne implementatsioon.

```
f={}
def fibonacci_dp_up_down(index):
    global f
    if index == 0 or index == 1:
        return index
    left = f[index - 1] if index - 1 in f.keys() else
       fibonacci_dp_up_down(index - 1)
    f[index - 1] = left
    right = f[index - 2] if index - 2 in f.keys() else
       fibonacci_dp_up_down(index - 2)
    f[index - 2] = right

    return left + right
```

Joonis 13. Fibonacci numbrite arvutamise DP-ga ülevalt-alla.

```
def fib_dp_down_up(index, f=[0, 1]):
    while index >= len(f):
        f.append(f[-1] + f[-2])
    return f[index]
```

Joonis 14. Fibonacci numbrite arvutamise DP-ga alt-üles.

II. Kasutatud koodi GitHub repositoorium

https://github.com/kare22/Methodology-for-evaluating-similarity-in-health-trajectories

III. Litsents

Lihtlitsents lõputöö reprodutseerimiseks ja üldsusele kättesaadavaks tegemiseks

Mina, **Karel Paan**,

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) minu loodud teose

   Haigustrajektooride sarnasuse hindamise metoodika,

   mille juhendaja on prof. Jaak Vilo,

   reprodutseerimiseks eesmärgiga seda säilitada, sealhulgas lisada digitaalarhiivi DSpace kuni autoriõiguse kehtivuse lõppemiseni.

2. Annan Tartu Ülikoolile loa teha punktis 1 nimetatud teos üldsusele kättesaadavaks Tartu Ülikooli veebikeskkonna, sealhulgas digitaalarhiivi DSpace kaudu Creative Commonsi litsentsiga CC BY NC ND 3.0, mis lubab autorile viidates teost reprodutseerida, levitada ja üldsusele suunata ning keelab luua tuletatud teost ja kasutada teost ärieesmärgil, kuni autoriõiguse kehtivuse lõppemiseni.

3. Olen teadlik, et punktides 1 ja 2 nimetatud õigused jäävad alles ka autorile.

4. Kinnitan, et lihtlitsentsi andmisega ei riku ma teiste isikute intellektuaalomandi ega isikuandmete kaitse õigusaktidest tulenevaid õigusi.

Karel Paan
03.08.2021