TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Tarkvarateaduse instituut

Siim Kurvet 123657IAPB

MEDITSIINILISTE ANDMESTIKU KORRASTAMINE JA MUSTRITE LEIDMINE

Bakalaureusetöö

Juhendaja: Martin Rebane
MSc

Tallinn 2017

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Siim Kurvet

22.05.2017

Annotatsioon

Käesoleva lõputöö eesmärgiks on suurte ja mahukate meditsiiniliste andmete viimine masinõpitavale kujule, nii et nende peal saaks rakendada andmekaeve algoritme. Nende abil on võimalik leida suurest andmehulgast seoseid ja erinevusi erinevate meditsiinilise ajalooga patsientide vahel, mille inimjõul leidmine oleks märksa keerulisem ning aeganõudvam.

Töö põhiprobleemiks on hulga erinevate andmetulpade viimine binaarsele kujule, mida nõuavad peamised seoseid otsivaid masinõppe algoritmid. Andmetulpade sisu varieerub, koosnedes nii arvudest, sõnedest, kuupäevadest, jah-ei tunnustest ning ka hulganisti tühjadest andmeväljadest.

Töö tulemuseks on korrastatud andmestik binaarkujul, mida saab kasutada sisendina levinumates masinõppe algoritmides, millega saab leida seoseid erinevate ravikäikude ja nende tulemuste vahel.

Lõputöö on kirjutatud Eesti keeles ning sisaldab teksti 27 leheküljel, 4 peatükki, 4 joonist, 9 tabelit.

Abstract

Preprocessing and pattern mining from medical dataset

The goal of current thesis is to preprocess existing big medical dataset to machine readable format so it is possible to use pattern mining algorithms on it. Those algorithms should help us in finding patterns and differences between patients with distinct medical history that otherwise might be unnoticed or ignored due to human error.

The main problem for this is transforming huge groups of different data columns to binary state which is required by the pattern mining algorithms. Original columns have dissimilar values like numbers, strings, dates, yes-no relations and great amount of missing data fields.

The result is properly preprocessed medical dataset, which fits as suitable input for pattern mining algorithms. Those algorithms are used to find frequent items sets from different type of preprocessed datasets and found results are analyzed to find differences and similarities between them.

The thesis is in Estonian and contains 27 pages of text, 4 chapters, 4 figures, 9 tables.

Lühendite ja mõistete sõnastik

CSV	*Comma Separated Values* Komaeraldusega väärtused – Porditav failivorming, kus andmebaasikirjed on üksteisest eraldatud komadega. Selles vormingus on iga rida üks kirje, mille väljad on üksteisest komadega eraldatud. Komade järel võib olla suvaline arv tühikuid ja/või tabeldusmärke (tab character), sest neid ignoreeritakse. Kui väli ise sisaldab koma, siis peab kogu väli olema ümbritsetud jutumärkidega [7] .
FP-Growth	*frequent pattern growth* Sagedalt esinevaid mustreid otsiv andmekaeve algoritm [6] .
ODS	*OpenDocument Spreadsheet* OpenDocument-vormingute faililaiend tabelarvutusteks [1] .
OpenDocument	*Open Document Format for Office Applications, "avatud dokumendivorming kontorirakendustele"* XMLi-põhine failivorming elektrooniliste dokumentide (nt tekstide, arvutustabelite, esitluste, diagrammide) jaoks [1] .
tab	Teksti fail, kus andmed on esitatud veergudena eraldatuna üksteisest tühja sõnega [4] .
XOR	välistav VÕI Loogikatehe, mille tulem on tõene, kui ükskõik kumb operand on tõene, kuid mitte mõlemad korraga [8] .

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Meditsiini arenguga läbi ajaloo oleme jõudnud punkti, kus erinevaid protsetuure ja ravimeetodeid on hulganisti. Nendega kaasnev info koos patsiendi eluteega annab kokku suure andmehulga. Üksikute ravilugude uurimine ja võrdlemine annab professionaalidele suure hulga vajalikku infot, aga suur patsiendite hulk, kus iga patsiendi kohta on ligi tuhat kirjet annab meile rohkem andmeid, mida inimmõistus suudaks korralikult töödelda.

1.1 Taust ja probleem

Taoliste käsiteldamatute andmehulkade puhul on loomulik pöörduda masinate poole ning rakendada nendel sagedasi andmehulki kaevandavaid masinõppe algoritme, mis aitaksid leida selliseid seoseid, mis inimesel muidu märkamatuks jääksid.

Taolisel lähenemisel on aga üks väga suur probleem. Nimelt on andmestikud üldjuhul loodud inimloetavana, arvestamata masinaga, mis sooviks binaarseid andmeridu. Kui eksisteeriv info hõlmab erinevaid andmetüüpe nagu kuupäevad ja tekstistringid, mida ei saa lihtsalt teisendada jah-ei sõltuvusteks, siis on vaja viia eksisteeriv andmete hulk oma algesest seisust binaarsete tunnusteni, mida saaks vastavale algoritmile ette anda. Kõik algoritmid ei vaja tingimata binaarkujul andmeid, aga vajadus tuleb konkreetse meditsiinilise andmestiku eripäradest, kus on otstarbekas selliseid algoritme rakendada.

1.2 Ülesandepüstitus

Käesoleva lõputöö eesmärgiks on korrastada eksisteerivat operatsiooni patsientide andmestikku binaarkujule mis sobiks sisendiks sagedaid andmehulki kaevandavale algoritmile. Korrastatud andmestikul rakendada FP-growth algoritmi ning analüüsida saadud tulemusi.

1.3 Metoodika

Algne andmekogu on 152e operatsiooni läbinud patsiendi andmestik ligi 850 sisendiga patsiendi kohta. Andmekogu korrastamiseks luuakse algoritm programmeerimise keeles Java, mis loeb sisse andmestiku ods (*OpenDocument Spreadsheet*) failina ja väljastab korrektse andmestiku tab failina. Korrastatud andmekogu peal rakendatakse Christian Borgelti FP-growth algoritmi [5] .

1.4 Ülevaade tööst

Teises peatükis käsitleb autor meditsiinilise algandmestiku korrastamise struktuuri, tekkinud probleeme ning kuidas nende lahendamisele läheneti, mida õpiti ja kuidas probleemidest üle saadi. Samuti analüüsitakse algandmestikku ning korrastamise kasu ning lõpptulemust.

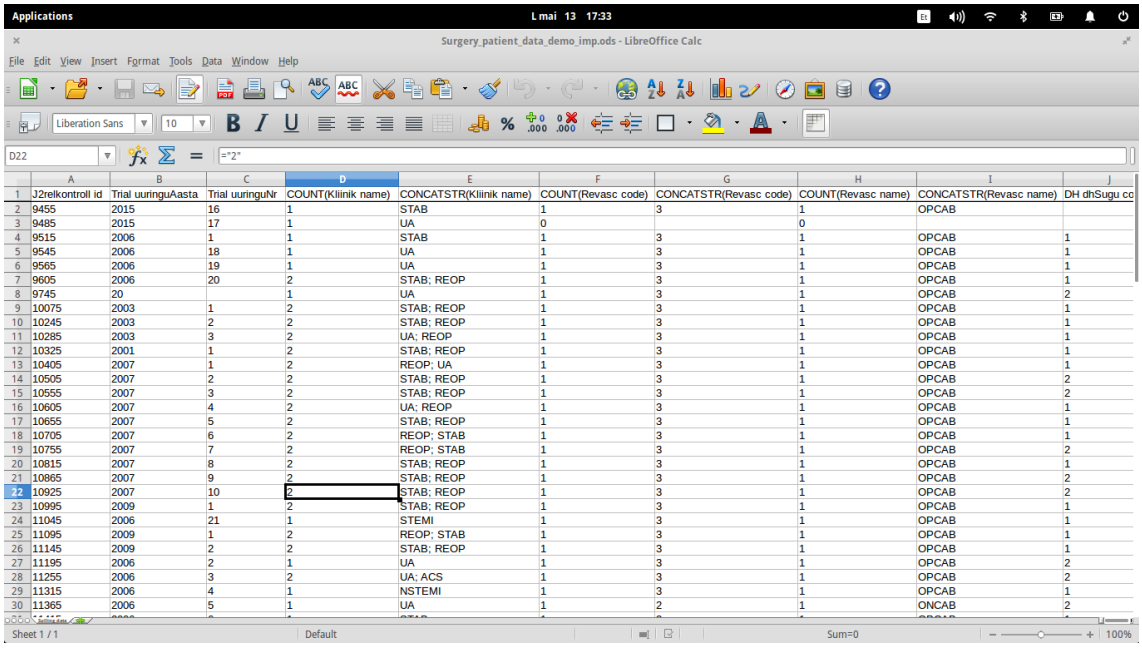
Kolmas peatükk räägib kasutatud andmehulkade kaeve algoritmidest lähemalt, kirjeldab saadud tulemusi koos nende omavahelise võrdluse ja analüüsiga.

2 Meditsiinilise andmestiku korrastamine

Alljärgnev peatükk kirjeldab meditsiinilise andmestiku korrastamiseks loodud algoritmi, analüüsides algse andmestiku peamisi probleeme selle binaarseks muutmisel ning nendest üle saamiseks välja töötatud lahendusi.

2.1 Algandmestiku kirjeldus

Algsed andmed on *.ods* failivormingus, mis on OpenDocument-vormingu faililaiend tabelarvutusteks [1] . Andmestik on avatav näiteks Microsoft Excelis, LibreOffice calc-is või mõnes muus OpenDocument toega tarkvararakenduses. Joonisel 1 on näidatud avatud algandmestik programmis LibreOffice Calc.



	A	B	C	D	E	F	G	H	I	J
1	J2relkontroll id	Trial uuringuAasta	Trial uuringuNr	COUNT(Kliinik name)	CONCATSTR(Kliinik name)	COUNT(Revasc code)	CONCATSTR(Revasc code)	COUNT(Revasc name)	CONCATSTR(Revasc name)	DH dhSugu co
2	9455	2015	16	1	STAB	1	3	1	OPCAB	
3	9485	2015	17	1	UA	0		0		
4	9515	2006	1	1	STAB	1	3	1	OPCAB	1
5	9545	2006	18	1	UA	1	3	1	OPCAB	1
6	9565	2006	19	1	UA	1	3	1	OPCAB	1
7	9605	2006	20	2	STAB; REOP	1	3	1	OPCAB	1
8	9745	20		1	UA	1	3	1	OPCAB	2
9	10075	2003	1	2	STAB; REOP	1	3	1	OPCAB	1
10	10245	2003	2	2	STAB; REOP	1	3	1	OPCAB	1
11	10285	2003	3	2	UA; REOP	1	3	1	OPCAB	1
12	10325	2001	1	2	STAB; REOP	1	3	1	OPCAB	1
13	10405	2007	1	2	REOP; UA	1	3	1	OPCAB	1
14	10505	2007	2	2	STAB; REOP	1	3	1	OPCAB	2
15	10555	2007	3	2	STAB; REOP	1	3	1	OPCAB	2
16	10605	2007	4	2	UA; REOP	1	3	1	OPCAB	1
17	10655	2007	5	2	STAB; REOP	1	3	1	OPCAB	1
18	10705	2007	6	2	REOP; STAB	1	3	1	OPCAB	1
19	10755	2007	7	2	REOP; STAB	1	3	1	OPCAB	2
20	10815	2007	8	2	STAB; REOP	1	3	1	OPCAB	1
21	10865	2007	9	2	STAB; REOP	1	3	1	OPCAB	2
22	10925	2007	10	2	STAB; REOP	1	3	1	OPCAB	2
23	10995	2009	1	2	STAB; REOP	1	3	1	OPCAB	1
24	11045	2006	21	1	STEMI	1	3	1	OPCAB	1
25	11095	2009	1	2	REOP; STAB	1	3	1	OPCAB	1
26	11145	2009	2	2	STAB; REOP	1	3	1	OPCAB	1
27	11195	2006	2	1	UA	1	3	1	OPCAB	2
28	11255	2006	3	2	UA; ACS	1	3	1	OPCAB	2
29	11315	2006	4	1	NSTEMI	1	3	1	OPCAB	1
30	11365	2006	5	1	UA	1	2	1	ONCAB	2

Joonis 1. Algandmestik avatuna LibreOffice Calc-is.

Lõputööks kasutatud algandmestikus on kokku 153 rida ja 849 veergu. Ridade arv miinus 1 on võrdne patsientide arvuga ning iga veerg kindla sissekandega antud

patsiendi kohta. Andmeväljade sisuks on nii sõned, sõnade loetelud, arvud, kuupäevad, kellaajad, kui ka kombinatsioonid eelnevalt mainitudest.

2.2 Algandmestiku sisselugemine

Andmete korrastamiseks on esmalt vaja need sisse lugeda. Kuna autori poolt sai programmeerimise keeleks valitud Java, siis sobis antud ülesandeks kõige paremini jOpenDocument teek. See on avatud lähtekoodiga Java teek, mis võimaldab lugeda, töödelda ja kirjutada OpenDocument formaadis faile [2] .

Loodud algoritmi puhul sai jOpenDocument-i võimalustest kasutatud ainult väikest osa. Täpsemalt algandmestiku ods faili sisselugemist ning selle ridade ning veergude kaupa läbikäimist. Joonisel 2 on kujutatud lihtsustatud näide ods faili sisselugemisest, mille põhjal autor realiseeris algandmestiku lugemise ja töötlemise [3] .

```
public void readODS(File file) {
	Sheet sheet;
	try {
		sheet = SpreadSheet.createFromFile(file).getSheet(0);
		int nColCount = sheet.getColumnCount();
		int nRowCount = sheet.getRowCount();
		MutableCell cell = null;
		for(int nRowIndex = 0;nRowIndex < nRowCount;nRowIndex++){
			int nColIndex = 0;
			for( ;nColIndex < nColCount; nColIndex++) {
				cell = sheet.getCellAt(nColIndex, nRowIndex);
				System.out.print(cell.getValue()+ " ");
			}
			System.out.println();
		}
	} catch (IOException e) {
		e.printStackTrace();
	}
}
```

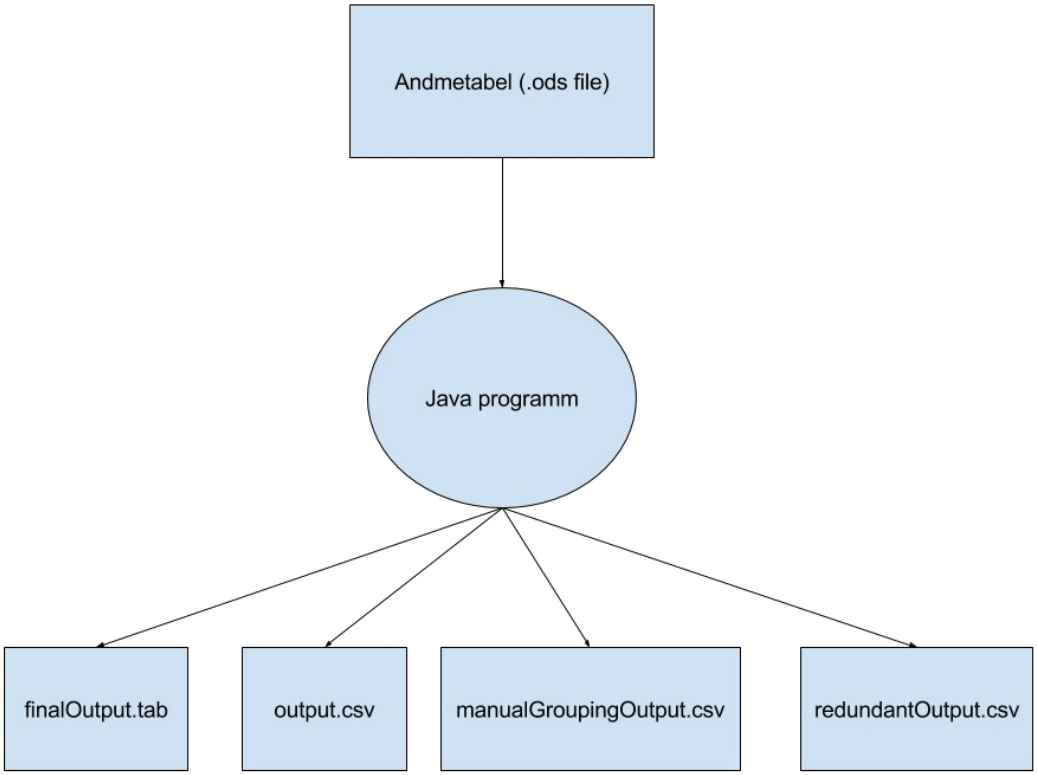
Joonis 2. Koodi näide jOpenDocumendi kasutatud osade toimimisest [3] .

Lahtiseletatuna loob eeltoodud programmikood sisendina antud ods faili põhjal '*Sheet*' tüüpi objekti. Saadud objekt kujutab endast andme tabeli maatriks tüüpi kirjeldust läbi ridade ja veergude. Iga tabeli ruudu, ehk '*MutableCell*' objekti saab kätte vastavalt tema rea ja veeru indeksile, mis võimaldab lihtsalt ning korduvalt vaatata kõiki tabeli väärtuseid.

2.3 Väljundfailid

Kuna algoritmi eesmärk on sisestatud andmetabel korrastada, siis programmi käivitamisega saame algsest OpenDocument failist andmehulkade kaeve algoritmile sobivama sisendfaili. Täpsemalt väljastatakse 3 csv (*Comma Separated Values*) faili: output.csv, manualGroupingOutput.csv ja redundantOutput.csv ja 1 tab vormingus fail: finalOutput.tab.

Neist esimene (output.csv) on korrastatud csv fail, mis kujutab binaariseeritud andmestiku ning on inimesele lihtsalt loetavas csv kujus, kus iga rida on üks lõplikest andmeobjektidest. Iga rea esimene väärtus on andmeobjekti tunnus, millele järgneb binaarne jada, kus on patsientide hulgale vastav arv elemente. Ainus tab vormingus väljastatav fail (finalOut.tab) omab sama sisu, mis (output.csv), aga on kasutatavale andmehulkade kaeve algoritmile sobivaimas formaadis, kus andmed on tabeli kujul ja väärtuste eraldajaks on koma asemel tühi tähemärk. Kolmas fail (manualGroupingOutput.csv) koosneb mõnest üksikust veerust mis on liialt erandlik üldiseks binaariseerimiseks ning nõuab eraldi käsitsi korrastamist või kõrvale jätmist. Neljas ning viimane fail (redundantOutput.csv) hõlmab endas kõiki andmeveerge, mis on üleliigsed. Need veerud on kas duplitseeritud mõne teise veeru poolt, liialt tühjad või omavad konstantset tunnust. Järgneval Joonisel 3 on kujutatud kogu sisend ja väljund andmete struktuur.



Joonis 3. Sisend ja väljund andmetestruktuur.

2.4 Duplikaatveerud

Algandmestiku üks probleemidest on duplikaatinfo, mis tuleneb peamiselt vajadusest osa infot esitada nii selle koodi kui ka reaalse väärtusega. Näiteks on kasutatud andmestikus olemas veerg 'DH dhSugu code', kui ka 'DH dhSugu name'. Kus esimene väljendab patsiendi sugu kas numbriga 1 või 2 ning teine annab soo tekstina, ehk kas 'mees' või 'naine'. Kusjuures kahe veeru vahel kehtib range sõltuvus, kus 1=mees ja 2=naine. Taolised duplikaatveerud aga raskendavad reeglisüsteemi kaeve algoritmide tööd, kuna see on automaatne üks ühele seos, mis suurendab juba niigi suurt lõppreeglite hulka.

Kuna järjestikku asuvad duplikaatveerud on vaja eemaldada, siis teeb algoritm iga uue rea sisselugemisel kontrolli, et ega järgnev veerg ei oma üks ühele seost hetkel loetava veeruga. Duplikaatveeru leidmisel lisatakse see 'redudantOutput' faili ning liigutakse järgmise veeru juurde.

Veergude omavahel võrdlemisel on oluline teha duplikaadi kontroll mõlemat pidi. Nii veerg A veeruga B, kui ka veerg B veeruga A. Kui teha kontroll ainult ühtepidi, näiteks ainult veerg A veeruga B, siis võib tekkida olukord kus A veerg omab unikaalseid väärtusi, samas kui veerg B omab ainult 2-3 erinevat väärtust, aga teostades kontrolli ainult veeru A vaatepunktist tekib olukord, kus igale veeru väljale vastab kindel veeru B väärtus ning antud veerg tunduks kui duplikaat.

Lisaks tuleb arvestada ka olukorraga, kus mõlemad veerud omavad ainult unikaalseid tunnuseid, mis jätab mulje duplikaatsetest ridadest. Seega tuleb kontrollida ka, et seoste leidmisel esineks vähemalt 1 seos rohkem kui ühe korra.

2.5 Puuduvad väärtused

Kuna andmestikud tekivad hulgast inimeste pool tehtud sissekannetest, siis on naturaalne et üheks probleemiks taoliste andmete juures on osa info puudumine. See tekitab probleemi, kus juhul kui mõnes veerus on andmeid sisestatud väga vähesel määral, siis kaotab see veerg oma väärtuse infoallikana. Puuduvate väärtuste puhul ei ole kindlalt teada, kas antud tunnus puudub või reaalsuses eksisteerib, kuid on jäänud lihtsalt märkimata.

Vähendamaks taoliste puuduvate andmeväljade mõju tasuks need lõppväljundist eemaldada. Kuna aga kindlapiirilist protsenti millest alates võiks veeru üleliigseks kuulutada pole teada, siis on kõige mõistlikum jätta see vabalt seadistatavaks nii, et vajadusel saaks liiga tühjade veergude leidmist karmistada või kergendada.

Loodud korrastamisalgoritmis on väljade tühjuse protsent defineeritud *config.properties* failis kui säte *minExistingDataPerCent,* mille väärtuseks on 0-100 arv, mis määrab ära kui palju täidetud väljasid peab ühes veerus vähemalt eksisteerima, et antud veerg lõppväljundisse lisada. Liiga tühjad veerud pannakse *redundantOutput* faili.

Teine probleem, mida tühjad veerud endaga kaasa toovad on nende märkimine lõppväljundis. Olenevalt kasutatavast andmekaeve algoritmist võib olla oluline märkida tühjad veerud selgesti eristatavana. Antud lõputöös kasutatav FP-Growth algoritm võtab 0e kui mitte eksisteerivaid väärtusi, seega selle jaoks kasutatud korrastus algoritmil võib

tühjad veeru väärtused märkida kui 0-id, kuna neid loetakse mitte eksisteerivateks tunnusteks.

2.6 Ühe väärtusega ja XOR veerud

Pealtnäha kõige otsekohesemad veergudest on sellised, mis omavad ainult ühte unikaalset väärtust. Nende lahendamine ideaalse andmestiku korral, kus ei ole mitte ühtegi puuduvat infovälja on äärmiselt lihtne. Nimelt kui kõik patsiendid omavad täpselt sama tunnust, näiteks on kõik läbinud mingi kindla protsetuuri, siis võib selle korrastamise käigus julgelt eemaldada, kuna antud tunnus on pidev ja ei anna mingit lisainformatsiooni. Pigem raskendab see algoritmi tööd ja suurendab selle väljundit tarbetult.

Mitteideaalne andmestik, nagu antud lõputöö tarbeks kasutatu, tekitas aga probleemi, kus veerus eksisteerib kas 1 tunnus või on info puudu. Taoliste veergude puhul tekib küsimus, kas puuduv info on teadlikult puudu või on lihtsalt märkimata jäänud, ehk kas puuduv väli on reaalsuses võrdne binaarse nulliga, ehk tunnuse puudumisega või on andmete täitmisel tehtud inimviga.

Korrastamise algoritmi raames lepiti algoritmi lihtsuse huvides FP-Growth tõlgendamise loogikaga, et tühi andmeväli tähendab tunnuse puudumist ning see on teadlikult märkimata jäänud. Seega kõik taolised veerud korrastatakse binaarselt, kus 1 tähendab, et tunnus eksisteerib ükskõik kas selle väärtuseks on sõne, number või kuupäev ja 0 märgib tunnuse puudumist.

2.7 Binaarsed veerud

Algandmestiku veergudest ühed lihtsaimad on ainult 2 erineva väärtusega veerud, kus ei ole ühtegi puuduvat väärtust. Taoliste veergude puhul tuleb ainult määrata milline tunnus on 1 ja milline 0 ning vastavalt sellele kogu veerg korrastada.

Keerulisem juht on kui lisaks kahele erinevale väärtusele eksisteerib ka puuduvaid andmevälju. See tekitab problemaatilise olukorra algoritmidele nagu FP-Growth mis arvestab puuduvate väärtustega kui binaarse nulliga, mis takistab antud veeru otsest

binaariseerimist, sest nagu jaotises 2.5 juttu, ei ole puuduvad väärtused reaalsuses ei üks ega teine tunnus. Näiteks võib taaskord kasutada andmeveergu 'DH dhSugu name', mis näitab patsiendi sugu, täpsemalt kas tegemist on mehe või naisega. Osade patsientide kohta aga selles veerus sooline info puudub ning tühje veerge mitte arvestades oleks tegemist juba justkui binaarse veeruga. Reaalsuses aga algoritmid nagu FP-Growth, mis ei arvesta puuduva väärtusega ning loevad selle tunnuse puudumiseks, tekitaksid taoliste veergude puhul, nagu näitena toodud sugu, ebakõla, mis võtaks kõiki puuduvaid väärtuseid kas mees või naine sõltuvalt kumb on märgitud kui 0 väärtus. Näiteks kui mees on märgitud kui 0 ja naine on märgitud 1ga, siis sel juhul arvestaks algoritm kõik puuduva sooga patsiendid meesteks, kuigi reaalsus võib olla vastupidine.

Taoliste 0, 1 või puudu tüüpi veergude probleemist mööda saamiseks on kõige lihtsam teha veerg kaheks eri väljundiks. Jäädes soolise veeru näite juurde, siis saab teha soo veeru vastavalt 'on patsient mees' ja 'on patsient naine' veergudeks. Kus kõik meessoost patsiendid märgitakse 1ga esimeses ja 0ga teises ning kõik naised täpselt vastupidi. Puuduvad väärtused jäävad aga nulliks, ehk puuduvaks väärtuseks mõlemas uues veerus.

2.8 Väikese hulga erinevate väärtustega veerud

Unikaalsete tunnuste arvu suurenemine veeru kohta raskendab ka nende binaariseerimist. Üle 2 tunnuse tähendab juba et veergu tuleb hakata jaotama vastavalt selle väärtustele. Samas kui erinevaid väärtusi on ainult väikene hulk, siis saab need lihtsalt jaotada sisendite põhjal uuteks binaarveergudeks, kus vastavalt igale patsendile on märgitud talle olemas olevad tunnused ühega ja puuduvad nulliga.

Näiteks on algandmestikus veerg 'DH dhRahvus name', millel on 3 eri väärtust: eestlane, venelane ja muulane. Korrastamise käigus luuakse sellest veerust 3 uut veergu rahvus_eestlane, rahvus_venelane ja rahvus_muulane ning täidetakse vastavalt iga patsiendi rahvusele märkides 1ga see uutest veergudest mille alla ta kuulub. Patsientide puhul kellel antud info puudub, märgitakse igasse loodud veergu väärtus 0.

Kui taolisete väikeste hulkade alampiir on kindla piiriliselt 3 unikaalset väärtust veeru kohta, siis ülempiir on lahtine ja sõltub suuresti olukorrast. Seetõttu on loodud

korrastusalgoritmis väikeste hulkade ülempiir seadistatav config.properties failis kui väärtus *maxNumberOfGroups.* Selle muutuja väärtusega saab määrata ära arvuliselt väikeste hulkade maksimaalse unikaalsete väärtuste arvu. Tühje väärtusi antud piiri otsimisel veerust ei arvestata, kuna puuduva väärtuse kohta uut binaarveergu nagunii ei looda.

Korrastusalgoritmi loomise käigus oli antud seadistuse väärtuseks algselt 5, aga algoritmi katsetuste käigus uurides algandmestiku, leiti antud piiri sobivamaks väärtuseks 6, mis kujutas endast peamist murdepunkti väikese hulga tihedate väärtustega veergude ja suure hulga hajusate väärtustega hulkade vahel. Kirjeldatud piiri optimaalse punkti leidmine on võrdlemisi vajalik, kuna suuremate hulkade puhul on oluline nende korrektne grupeerimine, mis paljude just taoliste 6 väärtusega veergude puhul oleks raskendatud. Alljärgnev joonis 4 demonstreerib programmikoodi millega on korrastatud väikese hulga erinevate sisenditega veerud.

```
private List<List<String>> correctMultipleValuesColumn(
      Map<String, Integer> values, int colIndex){
        Map<String, int[]> valueMap = new HashMap<>();
        for(String value : values.keySet()) {
            if(!value.equals("")) valueMap.put(value, new
                  int[datasheet.getRowCount() - 1]);
        }
        for(int nRowIndex = 1 ;nRowIndex < datasheet.getRowCount();
            nRowIndex++)
        {
            String cellValue = datasheet.getCellAt(colIndex,
                  nRowIndex).getValue().toString();
            if(!cellValue.equals("")) {
                valueMap.get(cellValue)[nRowIndex - 1] = 1;
            }
        }
        int count = 1;
        List<List<String>> newCols = new ArrayList<>();
        for (Object o : valueMap.entrySet()) {
            Map.Entry pair = (Map.Entry) o;
            List<String> colValues = new ArrayList<>();
            colValues.add("Multiple: " + datasheet.getCellAt(colIndex,
                  0).getValue().toString() + "_" + pair.getKey() + "_"
                  + count++);
            for (Integer i : (int[]) pair.getValue()) {
                colValues.add(i.toString());
            }
            newCols.add(colValues);
        }
        return newCols;
    }
```

Joonis 4. Väikese hulga unikaalsete väärtuste korrastamise programmikood.

Eeltoodud koodijupp väikese hulga väärtuste korrastamiseks võtab üheks sisendiks paisktabeli sõnedest ja täisarvudest. Sõned on korrastatavas veerus esinevad unikaalsed väärtused ja täisarvud väljendavad selle sõne esinemise sagedust antud veerus. Teiseks sisendiks on aga hetkel käsiteldava veeru number. Funktsioon loob esmalt uue paisktabeli sõnedest ja täisarvu jadadest. Tabelisse lisatakse iga unikaalse mittetühja väärtuse kohta tühi nullidest koosnev täisarvude jada. Seejärel käiakse läbi sisendina

antud veerunumbri põhjal algandmestikus vastav veerg rida-rida haaval, alustadest teisest reast, mis pole pealkirja rida. Iga rea juures leitakse korrektne välja väärtus ja otsitakse eelnevalt loodud tabelist leitud sisendile vastav täisarvude jada. Jadas muudetakse vastava reanumbriga element nullist üheks. Nõnda korra kogu veeru läbides saame tulemuseks paisktabeli kus igale unikaalsele väärtusele vastab nüüd binaararvude jada. Iga saadud jada ja sellele vastava tunnusvõtme paarist luuakse uus sõnede jada, mis on võrdne ühe uue veeruga lõppväljundis. Saadud nimekiri sellistest sõnede jadast tagastatakse väljundfaili printimiseks.

2.9 Suure hulga erinevate väärtustega veerud

Erinevatel patsientidel eksisteerivad väga erinevad sissekanded. Seetõttu leidub ka veerge kus erinevate väärtuste hulk on väga suur ja hajutatud. Selliseid veerge korrastades luua iga unikaalse tunnuse kohta uus veerg ja vastavalt patsiendile märkida kas ta omab seda tunnust või mitte paisutaks lõplikku korrastatud andmestiku liialt suureks. Näiteks kui meil on 150 realine andmestik, kus ühes veerus igal real oleks unikaalne tunnus, siis loodaks selle põhjal korrastamise käigus 150 uut veergu, kus igaühel oleks 1 positiivne väärtus ja 149 0-väärtust, mis ei oleks eriti ökonoomne ning raskendaks reeglikaeve algoritmi tööd kuna meil oleks hulk hõredaid reegleid, mis lõpptulemusena ei sobituks ühtegi sagedamini esinevasse andmehulka.

Antud probleemi lahenduseks on leida viis kuidas kirjeldatud suurt hulka unikaalseid andmeid grupeerida nõnda et saaks luua 3-5 tunnust mille vahel veeru väärtused saaksid ühtlaselt jaguneda. Nende 3-5 tunnuse põhjal saab luua korrastamisel uued veerud mille binaarsete väärtuste järgi on näha milline tunnus patsiendil eksisteerib ja milline mitte. Sellise jaotamise tulemusena saame lihtsalt korrastada veeru binaarkujule loomata ebavajalikult palju hajutatud veerge.

Kui veeru kõik võimalikud väärtused on olemas, siis on neid juba lihtne nullide ja ühtede jadaks teisendada. Keerulisem osa on aga algse suure hulga väärtuste sisukas grupeerimine. Jaotamiseks on oluline esmalt tuvastada kõigi väärtuste tüübid. Parimal juhul on kõik veeru väärtused numbrilised, sest numbrilisi väärtusi on lihtne järjestada kasvavas või kahanevas järjekorras ning vastavalt sellele gruppideks jagada.

Näiteks eksisteerib veerg 'DH dhVanus', mis hoiab endas iga patsiendi vanust ning kõik selle tunnuse väärtused on seega numbrilised. Alljärgnevas tabelis 1 on välja toodud 88 patsiendiga andmestikus esinenud vanuste veeru kõik unikaalsed väärtused ja nende esinemise sagedus.

Tabel 1. Patsientide vanused ja nende esinemise sagedus algandmestikus.

vanus	**Esinemise kordade arv**
33	1
48	1
49	1
52	1
53	1
54	4
55	2
56	4
57	1
58	1
59	2
60	4
61	3
63	1
64	1
65	2
66	4
67	2
68	8
69	4
70	5
71	4
72	1
73	4
74	3
75	4
76	4

vanus	Esinemise kordade arv
77	2
78	1
79	1
80	1
81	1
88	1
Vanus puudu	8

Nagu ülalolevast tabelist näha siis on algandmestikus 80 täidetud väärtust ja 8 puuduvat 88st. Täidetudest on unikaalseid vanuseid 33. Nagu mainitud, siis ühest vanuse veerust 33 veeru tegemine ei oleks otstarbekas. Loodud korrastus algoritmi eesmärgiks on taolise andmestiku juures luua 3-5 vanuste vahemikku, mis oleks võimalikult ühtlaselt jaotatud. Ühtlane jaotus on oluline selleks et vältida olukorda kus tekib üks grupp mis on teistest kordades suurem ning seega looks liiga ühekülgse jaotuse andmekaeve algoritmi tarbeks. Seega antud veeru grupeerimiseks tasuks leida optimaalne keskmine arv palju igas grupis elemente olla võiks. Selle tarbeks leiab algoritm täidetud väärtuste arvu, mis on 80 ning jagab selle arvu neljaga, mis on optimaalne gruppide arv ning liidab saadud arvule 1-e jagamises tulevate ümardus ebatäpsuste vältimiseks. Antud näite puhul on seega saadud mediaanväärtuseks 21. Vastavalt saadud väärtusele täidab algoritm gruppe alustades kõige esimesest elemendist ning 21-e täitudes või sellest üle minnes alustab uue grupi täitmist. Kusjuures mediaanväärtusest üle minnes võrreldakse kaugust mediaanväärtusest hetke grupi täituvusega et leida kas mediaani ületanud väärtus läheb antud grupi lõppu või alustab uut gruppi. Lõpetades vanuse veeruga saame tulemuseks tabelis 2 kuvatud jaotuse.

Tabel 2. Patsientide vanuste jaotus ja jaotuse elementide arv.

Vanuste vahemik	Vahemiku elementide arv
33-60	23
61-68	21
69-74	21

Vanuste vahemik	Vahemiku elementide arv
75-88	15

Nagu näha siis algoritmi tulemusena saame 4 võrdlemisi ühtlaselt jaotunud grupeeringut vanustest. Grupeeringute põhjal luuakse 4 uut veergu, mis täidetakse vastavalt ridade sisule märkides ühega kõik patsiendid kes kuuluvad antud veeru vanuste vahemikku.

Eelnevalt toodud näide kujutab endast aga parimat juhtu, kui arvu väärtused on numbrilised. Teisteks levinuimateks suure hulgaga andmeveergude tüüpideks on kuupäevad ja kellaajad. Mõlemaid oleks aastate või päeva lõikes samuti lihtne järjestada ja seega ka jaotada, aga sellisel grupeerimisel puuduks sügavam sisuline mõte, kuna suure tõenäosusega ei ole väga vahet mis kell mingi protseduur või mis kuupäev mõni operatsioon aset leidis. Seega ei ole mõtet kuupäevi, ega kellaegu grupeerida nagu numbrilisi jaotusi. Selle asemel tasub aga luua kuupäeva või kellaja veerust binaarne veerg kus 1 tähistab ajalise väärtuse eksisteerimist ja 0 aja puudumist antud patsiendi puhul. Sellise korrastamise põhjuseks oleks, et kuigi ajalisel grupeerimisel sisu puudub, siis kindla kuupäeva või kellaaja omamamine tähistab mingi protseduuri läbi viimise toimumist, mis väärib tunnusena märkimist.

Viimaseks ning kõige keerulisemaks veergude tüüpiks on aga suure hulgaga veerud, mille andmed on kas sõned või segu sõnedest ja numbridest. Viimase näiteks eksisteerib veerg 'OperatsioonAks oaDistIII' mille väärtusteks on 1-8 ja stringid 'SPL' ning 'x'. Ainult sõnede näiteks saaks tuua veeru 'Surm sp6hjus name', mis sisaldab erinevaid surmapõhjuseid nagu äkksurm, infarkt ja muud. Taoliste veergude grupeerimiseks aga pole kindlat meetodit kuna nende sisu pole lihtsalt järjestatav ja seega on võimatu leida ühiseid tunnuseid mis võimaldaksid luua neist iseseisvad sidusad jaotised. Kuna pea kõik taolised veerud on eraldiseisvad juhud, siis võib neid lugeda erandolukordadeks, mis nõuab inimlikku otsutust vastavalt vajadusele. Seega eraldab korrastus algoritm kõik taolised veerud ja kirjutab need üles 'manualGroupingOutput' faili.

2.10 Korrastatud andmestik numbrites

Kasutatud algandmestikus on 849 veergu ja 153 rida, ehk kokku 129897 andmevälja. Korrastus algoritmi tulemusena sai sellest 3 unikaalse sisuga faili. Korrastatud fail sobilik reeglikaeve algoritmile on lõpptulemusena 153 rida ja 544 veergu, mis on tulnud 229-st arvust originaalsetest veergudest. Üleliigsete sisendite failis on 607-e originaalsele algandmestikule vastavat veergu. Viimaks inimülevaatuseks mõeldud failist leiab 13 originaalset veergu mida algoritm ei suutnud iseseisvalt korrastada. Alljärgnev tabel 3 annab konkreetse ülevaate algandmestiku veergude jagunemisest olenevalt väljundfailist.

Tabel 3. Ülevaade originaalsete veergude jaotusest väljundfailides.

Faili nimi	**Kasutatud originaalse andmestiku veergude arv**	**Kasutatud veergude arvu protsent**
output.csv	229	27%
redundantOutput.csv	607	71.5%
manualGroupingOutput.csv	13	1.5%
Kokku	**849**	**100%**

Nagu ülalolevast tabelist näha, siis üle kahe kolmandiku, peaaegu kolm neljandiku algsetest andmedest on kasutatud seadistuse juures üleliigsed. Alljärgnes tabelis 4 on välja toodud kõikide üleliigsete veergude esinemise põhjused.

Tabel 4. Ülevaade üleliigsetest sisenditest ja nende tekke põhjustest.

Üleliigsuse põhjus	**Veergude arv üle-liigsede veergude failis**	**Veergude arv ilma duplikaatide otsin-gutta**	**Veergude arv ilma duplikaatide ja tühjuse kontrolliga**
duplikaatveerg	409	-	-
liialt tühi veerg	194	407	-
konstantsed väär-tused	4	65	235
Kokku	**607**	**472**	**235**

Esitatud jaotusest on võimalik välja lugeda et peamine üleliigsus on andmete kordumine algandmestikus. Need moodustavad kaks kolmandikku ebavajalikest sisenditest ning

pea pool kogu esmasest andmestikust. See tuleneb suuresti sellest et paljud andmed on duplitseeritud kui kood ja nimeline väärtus. Lõppfailis on liialt tühje veerge märgitud 194. See pole aga täielikult korrektne arv, sest veergude puhul sooritatakse esmalt duplikaadi kontroll. Seega selle eemaldades saame et algandmestikust 407, taaskord pea pool, veergudest on liialt tühjad. Sellest võime järeldada, et suures osas on tegemist võrdlemisi hajusa andmestikuga, kuna puudavate andmeväljade arv on võrdlemisi suur. See arv tuleb seadistatud nõudest et vähemalt 15% veerust peaks olema täidetud. Järgnevas tabelis 5 on välja toodud liialt tühjade veergude arv muutes täidetuse nõuet.

Tabel 5. Veeru täidetuse protsent vastavuses liialt tühjade veergude arvuga.

Täidetuse nõue (%)	Liialt tühjade veergude arv
0	170
5	315
10	379
15	407
20	419
25	432

On näha et suurem osa tühjade veergude hulgast tekib juba kõigest 5 protsendilise täidetuse nõude juures ning 170 veergu on täiesti tühjad, mis annab taaskord kinnitust et antud meditsiinilised andmed on hõredalt täidetud ning sisaldavad hulga tunnuseid, mis on üleliia või esinevad ainult väga erandlikel juhtudel.

Pöördudes tagasi tabeli 4 juurde tasub süveneda ka üleliigsete konstantsete väärtuste numbritesse. Sealt on näha et pea kõigi konstantsete veergude puhul on tegemist kas juba duplitseeritud või liialt tühja veeruga. Viimane on ka mõistetav kuna konstantsete veergude alla lähevad ka kõik täiesti tühjad veerud. Duplitseerituse ja konstantsuse seos tuleneb aga suurest hulgast järjestikku olevatest hinnangutest, mis tunduvad omavat sama vaikeväärtust mida andmestiku hõreduse tõttu pole muudetud.

Võttes kokku kogu üleliigsete andmete statistika on selgelt näha korrastus algoritmi suurt kasu, sest see suudab eemaldada andmestikust 70% üleliigseid andmeid, mida muidu oleks pidanud korrastama käsitsi. Juba ainuüksi taoline algandmestiku puhastamine hoiab kokku palju aega ja vaeva.

Kuigi üleliigseid sisendeid on palju, siis leidub ka piisavalt sobilikke andmeveerge. Järgnevalt on tabelis 6 välja toodud kõikide sobivate veergude jaotus andmestikus vastavalt sellele mis moodi neid tuli korrastada.

Tabel 6. Algandmestiku sisuliste veergude jaotus vastavalt nende korrastamise liigile.

Veeru korrastamise liik	**Veergude arv algandmestikus**	**Veergude arv väljundfailis**
Binaarsed veerud	55	55
Väike hulk väärtusi	112	316
Suur hulk numbrilisi väärtusi	39	150
Suur hulk ajalisi väärtusi	23	23
Kokku	**229**	**544**

Uurides eelpool toodud tabelit hakkab esimesena silma et üle poole korrastamiseni jõudnud veergudest sisaldavad väikest hulka erinevaid väärtusi. Sedasi korrastatud veerge on lõppväljundis 316, mis on 2.82 korda rohkem kui kui originaalsete veergude arv. Arvestades et kasutatud seadete juures võib ühest taolisest veerust saada 2-6 uut veergu, siis võib järeldada et valdav enamus taolisi veerge moodustavad lõpptulemusena 2-3 uut veergu, ehk peamiselt on tegemist tunnustega millel on 2 unikaalset väärtust ja mingi hulk puuduvaid väärtusi.

Algsest 849-st veerust 55 on sellised millel sai rakendada binaarset korrastus loogikat, ehk kus oli kas ainult 2 unikaalset tunnust ilma puuduvate väärtusteta või üks unikaalne tunnus ja puuduvad väärtused. Andmestiku uurimine näitas et binaarsed veerud jagusid vastavalt 20 ja 35, ehk võrdlemisi ühtlaselt, kuigi puuduvate väärtuste ülekaal on taaskord märgatav. Binaarsete veergude suureks eeliseks on see, et neist ei teki lõppväljundisse lisaveerge, erinevalt teistest korrastustüüpidest peale ajaliste, mis teisenduvad samuti üks-ühele.

Suure hulgaga erinevate väärtustega veergude arv on summaarselt ligikaudu võrdne binaarsete veergudega algandmestikus, aga mõistetavalt loovad need lõppfaili hulganisti rohkem uusi veerge.

Vaadeldes tabelisse 6 kogutud üldpilti, siis keskeltläbi iga korrastamiseks sobiliku rea kohta luuakse 2,4 uut veergu. Parimal juhul oleks see arv 1 ja antud seadete puhul halvima juhul 6, kui kõik algsed veerud koosneksid kuuest eri tunnusest. Samuti on näha korrastamise algoritmi vajalikust ka korrastamisel endal, kus hoolimata suurest kärpimisest on ikkagi palju andmeveerge mis nõuavad erinevat lähenemist ja produtseerivad suure hulga uusi veerge, mida käsitsi oleks märgatavalt raskem luua.

3 Korrastatud meditsiinilisest andmestikust mustrite leidmine

Omades nüüd binaarset andmestikku, mis sobib sisendiks masinõppe algoritmidele tuleb järgmise sammuna ka seda rakendada. Selleks kasutame masinõppe algoritmi mis aitaks leida korrastatud andmetest sagedasti esinevaid andmehulki. Kasutatavateks väärtusreegliteks on korrastamisel loodud veerud, kus iga veerg on 1 kirje mis patsiendil eksisteerib või mitte.

3.1 Andmehulkade kaeve algoritmid

Sagedasti esinevate andmehulkade kaeve algoritmid loodi esmalt kaupluste ja e-poodide poolt klientide sooritatud ostudes seaduspärasuste leidmiseks. Eesmärkiks oli identifitseerida tooted mida osteti tihti koos. Selle põhjal sai poes kaupa organiseerida ja soovitada vastavalt kliendi hetke ostukorvi sisule [5] . Hiljem on rakendatud taolisi algoritme ka muudel andmestikel ning algseid algoritme on edasi arendatud kiiremaks ja võimekamaks. Enimtuntud selle valdkonna algoritmid on Apriori, Eclat ja FP-Growth [6] . Antud lõputöö raames valiti kasutatavaks algoritmiks neist viimane.

3.2 FP-Growth

FP-Growth on pikemalt välja kirjutatult *frequent pattern growth [6]* . Antud algoritmi näol on Apriori algoritmi edasi arendusega, mille peamiste eelisteks nimetatu ees on kandidaatobjektide hulga mitte genereerimine. Nimelt loob see olemasolevatest andmedest hulga alamhulki, mis tuleb ka kõik samamoodi läbi käia kui algne andmestik [6] . FP-Growth käib aga algse andmestiku läbi vaid 2 korda. Nende kordadega loob algoritm kompaktse andmete struktuuri puu, kust algoritmi teises osas leitakse kõik otsitavad sagedased andmeobjektide hulgad [6] . Selline andmestiku kaevandamine on märksa efektiivsem, kui teistel sarnastel algoritmidel.

Käesoleva lõputöö raames kasutatakse Christian Borgelti poolt välja töötatud FP-Growth algoritmi implementatsiooni. Borgelt on välja töötanud käsurealt käivitatava FP-Growth algoritmi mis soovib sisendiks korrektset andmestiku, millel oleks võimalik andmekaevet teostada. Väljundiks on fail kõikide sagedasti esinevate andmehulkadega [5] .

3.3 Metoodika

Analüüsi teostamiseks jaotame algandmestiku 3 võrdseks osaks patsientide peamiste tunnuste põhjal. Nendest alamandmestikest leitakse andmekaeve algoritmi põhjal kõik sagedamini esinevad andmehulgad ja võrreldakse saadud tulemusi, otsides sarnasusi ja erinevusi.

Kasutatav andmestikus on kõik operatsiooni läbinud patsiendid, kellest saab luua 3 suuremat gruppi: surnud patsiendid, kordusoperatsioonil käinud elus patsiendid ja ülejäänud ühe operatsiooniga piirdunud isikud. Nagu alljärgnevast tabelist 7 selgub saab sellise jaotuse põhjal 3 võrdlemisi ühtlase suurusega gruppi mida omavahel võrrelda.

Tabel 7. Patsientide grupeerimine algandmestikus.

Patsiendi tüüp	**Patsientide arv algand-mestikus**
Surnud patsiendid	60
Kordusoperatsiooni läbinud elus patsiendid	50
Ühe operatsiooni läbinud elus patsiendid	42

Borgelti FP-Growth käsurea rakenduse kasutamiseks tuleb läbida 2 sammu. Esiteks tuleb jooksutada käsku *flg2set* korrastatud andmestiku faili peal, mis teisendab selle Borgelti algoritmile sobivasse andmete esituse tüüpi. Autori poolt loodud korrastatud fail on kujul kus iga rida on 1 patsient ning iga veerg väljendab ühte tunnust, mis patsiendi reas on märgitud kui 1 (eksisteerib) või 0 (ei eksisteeri). Käsuga *flg2set* aga asendatakse kõik 1-d reas selle veeru tunnusega esimesest reast ignoreerides 0-e. Nõnda tekib uus fail, kus on kadunud esimene rida veeru tunnustega ning iga rida omab nimekirja veeru tunnustest mis vastaval patsiendil olemas on [5] .

Teise sammuna rakendatakse esimeses sammus modifitseeritud andmetega FP-Growth algoritm. Selleks tuleb käsureal kasutada käsku: *„fpgrowth [sätted] sisendfail [väljundfail]”,* kus nurk sulgudes olevad väärtused on valikulised [6]. Sisendiks läheb eelnevalt modifitseeritud andmefail ning vastavalt antud sätetele kirjutatatakse väljundfaili kõik leitud sagedased andmeobjektid. Programmile sisendiks on võimalik anda hulga erinevaid sätteid, millega saab piirata või suurendada otsitavate andmete mahtu, modifitseerida väljundit ja palju muud [5] .

Antud lõputöö raames keskendume kahele peamisele seadistuse võimalusele. Esmalt on kasutusel säte *-s#*, mis võimaldab määrata minimaalset nõutud andmehulga esinemise sagedust. Trellide märgi asemel tuleb sisestada täisarvuline protsent, mis peab olema täidetud et andmehulk kuuluks sagedasti esinevate alla. Vaikeväärtus antud seadel on 10 protsenti [5] .

Teiseks kasutatavaks seadistuseks on *-n#*, mis laseb määrata maksimaalse arvu andmeobjekte ühes andmehulgas [5] . Vaikeväärtuseks sellel on piiramatu arv objekte, mille puhul aga praktika näitas, et antud suurust tuleb märgatavalt piirata, et väljundite hulk ei läheks lõpmata suureks.

3.4 Andmekaevanduse tulemused

Esmase seadena kaevandamisel sai kasutatud vaikeväärtusi s=10 ja n=piiramatu. Piiramatu objektide arv hulgas osutus aga kasutatud andmestiku puhul liiga suureks ja algoritmi töö käigus jäi kõvaketta ruumist väheks, mistõttu tuli kaevandamine katkestada.

Selleks, et lõppmahtu hoida normaalsuse piirides, tuleb piirväärtusi karmistada. Andes algoritmile andmehulkadesse kuuluvate objektide maksimaalse väärtuse jääb võimalikke andmeobjektide kombinatsioonide arv mõistlikuse piiridesse. Teise piiramisena saab tõsta minimaalse esinemise sageduse protsenti. Algväärtus 10 eeldab, et andmeobjektide hulk peab esinema kümnel protsendil patsientidest. See arv võib olla aga liiga madal arvestades kui väike arv patsiente on igas sisendanmestikus peale 3ks jaotamist. Ligi 50ne patsiendi puhul nõuab see andmeobjektide korduvat esinemist kõigest 5lt patsiendilt, mis on statistilises mõttes väga väike suurus. Tabelis 8 on välja

toodud kõik järeleproovitud seadistused erinevate patsiendi gruppide kohta ning saadud väljundfaili maht ja sisalduvate andmehulkade, ehk ridade arv.

Tabel 8. Ülevaade FP-Growth algoritmi kasutamisel saadud andmehulkade mahtudest.

Algoritmi seadistus	Patsiendi tüüp	Andmehulkade arv	Lõppfaili maht
-s10 -n3	surnud	2 256 011	385.1 MB
	Kordusoperatsioon (elus)	2 483 467	416.1 MB
	Üks operatsioon (elus)	692 893	113.9 MB
-s20 -n3	surnud	991 955	170.1 MB
	Kordusoperatsioon (elus)	1 269 642	215.5 MB
	Üks operatsioon (elus)	349 902	57.0 MB
-s40 -n3	surnud	392 510	67.9 MB
	Kordusoperatsioon (elus)	536 874	92.0 MB
	Üks operatsioon (elus)	91 539	15.2 MB
-s20 -n4	surnud	32 797 514	7.4 GB
	Kordusoperatsioon (elus)	46 877 349	10.6 GB
	Üks operatsioon (elus)	9 434 115	2.0 GB
-s40 -n4	surnud	11 058 806	2.5 GB
	Kordusoperatsioon (elus)	16 821 444	3.8 GB
	Üks operatsioon (elus)	1 411 879	309.6 MB

Ülalolevat tabelit 8 uurides võib järeldada, et peamiseks lõppväljundi suurendajaks kasutatud andmestiku puhul on lubatud andmeobjektide arv hulgas. Suurendades seda ühe võrra kolmelt neljale, suureneb lõppresultaat mitmekümnekordselt sama sagedus protsendi juures. Üldjuhul ulatub sellise seadistusega andmehulkade fail mitmetesse gigabaitidesse ning nende enda töötlemine nõuaks juba omakorda korralikumat töötlusalgoritmi.

3.5 Sagedaste andmehulkade võrdlus patsiendi tüüpide kohta

Võrreldes infot patsientide tüüpide vahel tabelis 8, siis eristub selgelt iga seadistuse puhul ühe operatsiooni läbinud elusate patsientide grupp, mille väljund andmehulkade arv ja faili suurus on kordades väiksemad kui teistel patsientide gruppidel. Sellest võib

järeldada, et algandmestikus enamus puuduvatest väärtustest kuuluvad just sellistele patsientidele. Selle põhjuseks on tõenäoliselt suur hulk andmeveerge, mis hoiavad kas infot patsiendi surma põhjuste või erinevate kordusoperatsiooniga seotud toimingute kohta.

Võrreldes surnud ja kordusoperatsiooni läbinud elusaid patsiente tasub esmalt tähele panna seda, et kuigi esimesi patsiente (60) on rohkem kui teisi (50), siis sagedasi andmehulki leidub rohkem kordusoperatsiooni läbinud patsientide puhul. Lisaks võib märgata tendentsi nende kahe grupi vahel minimaalse sagedus protsendi (-s) tõstmisel, kus 10% juures on surnute grupi andmehulk 90% kordusoperatsiooni hulkade arvust, kuid 40% juures on vastav näitaja 73%. See tuleneb sellest, et osa surnudest on samuti kordusoperatsioonil käinud patsiendid, kuid sagedus protsendi tõstmisel kõrgemale selliste surnute jaotuse piirist hakkab kordusoperatsiooniga seotud andmeobjektide hulk vähenema. Sellest võib järeldada et algandmestikus on kaalukalt enim infot patsientide kohta, kes on käinud korduvalt operatsioonidel. Nimelt on neid tõenäoliselt rohkem analüüsitud korduvate käimiste jooksul ja neil eksisteerib taoline teise operatsiooni info, mida seda mitte läbinud, omada ei saa.

Lisaks saadud sagedaste andmehulkade suurusjärgu numbritele tasuks uurida ka nende sisu. Selle tarbeks sai loodud Java programm, mis sisendiks võtab kahe sagedaste andmehulkade faili ning leiab neist mõlemas korduvad andmehulgad, sealhulgas arvestades andmehulga objektide permutatsioonidega. Võrdluseks kasutati FP-Growth algoritmi seadistusega '-s20 -n3' leitud sagedasi andmehulki, sest -n4 failid osutusid liiga suurteks ja kahekümne protsendiline sagedus on sobivaim arvestades kasutatud jaotatud andmestike väiksust. Tabelis 9 on välja toodud saadud andmehulkade ühisosa informatsioon.

Tabel 9. Eri patsiendi gruppide ühised sagedased andmehulgad.

Võrreldavad patsientide grupid	**Ühiseid andme-hulki**	**Maksimaalne ühiste andme-hulkade arv**	**Ühilduvuse protsent**
1 operatsioon (elus) – korduvoperatsioon (elus)	8346	349 902	2.4%
1 operatsioon (elus) - surnud	7849	349 902	2.2%
Korduvoperatsioon (elus) - surnud	529 776	991 955	53.4%

Vaadates tabelit 9 on taaskord näha väga suurt erinevust ühe operatsiooni läbinute ja kahe teise grupi patsientide vahel. Leitud sagedaste andmehulkade kattuvus on sellistel puhkudel üllatavalt madal. Kõigest ligikaudu 2.4%, korra operatsioon läbinute andmeobjektide hulgad, on esindatud ka teistes andmehulkades, mis märgib et operatsiooni kõigest korra läbinud patsientide info erineb karmima saatusega patsientide omast märgatavalt rohkem. Üllatavalt kombel aga korduvoperatsioonide ja surnud patsientide ühisosa on üle poole maksimaalsest võimalikust. Suurem osa sellest on ilmselt taaskord tingitud sellest, et surmaga lõppenud juhtumite hulgas on ka mitmeid kordusoperatsioonil käinud patsiente. Hoolimata sellest on siiski vahe ühe kordsete õnnelike patsientidega võrreldes märkimisväärne. Seega võib öelda et surnud ja kordusoperatsioonil käinud patsientidel, ehk problemaatilisematel juhtumitel on üksteisega rohkem ühist. See näitab et andmestikust on võimalik leida tunnuseid, mis viitavad sellele kui tõsise juhtumiga on tegemist ja kas tõenäolisem on see et patsient piirdub ühe operatsiooniga või tuleb operatsioone ning komplikatsioone rohkem ette.

3.6 Järeldused

Analüüsides kõiki FP-Growth kasutamisel saadud sageli esinevaid andmehulki võib teha mõndasi esialgseid järeldusi seosetest problemaatilisemate ja kergemate patsientide vahel. Taoliste seoste täpsemaks uurimiseks on vaja aga meditsiiniliste teadmistega süüvida sügavamale leitud andmehulkadesse. Selle töö lihtsustamiseks aga tuleks täiendada algandmestiku ja meditsiinilise pilguga üle vaatata korrastatud andmestik enne andmekaeve algoritmi kasutust.

Arvestades saadud tulemuste suurust, siis selle naturaalseks piiramiseks oleks vajalik korrastatud veergude piiramist sõltuvalt nende meditsiinilisest väärtusest, et eemaldada üleliigsed veerud, mida korrastus algoritm tunnuste nime põhjal ise otsustada ei suuda. Sellise piiramise tulemusel annaks hulgaliselt limiteerida lõppväljundite hulka ning võimalusel ühe andmehulga maksimaalsete objektide suurust muuta.

Samuti oleks kasulik suurendada valimi hulka, mis aitaks ühtlustada puuduvast infost tulenevaid ebavõrdsusi lõppväljundis. Antud juhul algoritmiga läbi töötatud

maksimaalselt 60 realised andmestikud ei tekitanud ühtegi jõudlus probleemi, seega ruumi andmete lisamiseks rea vaates on enam kui küll.

Kuigi genereeritud andmemahud tunduvad läbivaatamiseks väga suured, siis valdkonna eksperte kaasates saab täpsustada uuritavad tunnused ja praegu genereeritud mustrite (*frequent pattern*) pealt üle minna assotsiatsioonireeglite peale (st ütleme ette, millise tunnuse või tunnuste grupi iseloomustamise jaoks mustreid soovime). Need 2 sammu koos kitsendavad programmi tulemi inimesele hoomatavale kujule.

4 Kokkuvõte

Lõputöö põhieesmärgiks oli viia andmebaasi tüüpi meditsiinilise infoga andmestik andmekaevandus algoritmidele sobivale binaarkujule ja saadud väljundi peal rakendada sageli esinevate andmehulkade kaeve algoritmi ning selle tulemusi analüüsida.

Andmete korrastamiseks loodi programmeerimis keeles Java korrastus algoritm mis loeks sisse ods failivormingus andmete tabeli ja teisendaks seal leiduvad andmed vastavateks binaarseteks väärtusteks. Korrastamisel lahati algandmestikus esinevaid tüüpilisi probleeme nagu duplikaatväärtused ning nende eemaldamine, puuduvate väärtustega arvestamine, väikese hulga erinevate väärtuste sorteerimine ning suure hulga unikaalsete väärtuste ühtlasteks gruppideks jagamine vastavalt nende sisule. Korrastamise tulemusena kahandati algset andmestikku märgatavalt eemaldades kaevandamiseks ebavajalikud andmeveerud.

Algoritmi poolt korrastatud andmestik jagati patsendi tüüpide põhjal kolmeks võrdseks osaks ning kasutades FP-Growth algoritmi leiti iga jaotuse põhjal sagedasti esinevad andmeobjektide hulgad.

Töö tulemusena võib järeldada, et meditsiiniliste andmestikke on võimalik viia sellisele masinloetavale kujule, mis võimaldab meil selle peal kasutada andmekaeve algoritme, millega on võimalik leida sagedasti esinevaid andmehulki ja muid seoseid, mis võivad tulevikus aitata paremini diagnoosida ja ennetada meditsiinilisi probleeme arstiabi vajatel inimestel. Samuti selgus töö käigus, et paremate tulemuste saamiseks on oluline olemasoleva algandmestiku valimi suurendamine ja koostöö meditsiiniliste organisatsioonidega korrastamisel meditsiiniliselt ebatähtsate väärtuste eemaldamiseks.

Kasutatud kirjandus

[1] OpenDocument. [WWW] https://et.wikipedia.org/wiki/OpenDocument (13.05.2017)

[2] jOpenDocument. [WWW] http://www.jopendocument.org/ (13.05.2017)

[3] Read OpenOffice SpreadSheet (ods) [WWW] http://half-wit4u.blogspot.com.ee/2011/05/read-openoffice-spreadsheet-ods.html (13.05.2017)

[4] .tab File Extension [WWW] https://fileinfo.com/extension/tab (19.05.2017)

[5] FP-Growth. Find Frequent Item Sets with the FP-growth Algorithm [WWW] http://www.borgelt.net/doc/fpgrowth/fpgrowth.html (19.05.2017)

[6] Association rule learning. [WWW] https://en.wikipedia.org/wiki/Association_rule_learning (19.05.2017)

[7] H. Vallaste, „CSV (Comma Separated Values)“ [WWW] http://www.vallaste.ee/ (21.05.2017)

[8] H. Vallaste, „XOR“ [WWW] http://www.vallaste.ee/ (22.05.2017)

Lisa 1 – Kood Git'i repositooriumis

https://bitbucket.org/siim_kurvet/thesis.git