TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Informaatikainstituut
Tarkvaratehnika õppetool

Inforikaste reeglisüsteemide leidmine binaarandmestike jaoks

Bakalaureusetöö

Üliõpilane: Hendrik Kivi
Üliõpilaskood: 134928IAPB
Juhendaja: Martin Rebane

Tallinn
2016

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.

---------- ----------

(*kuupäev*) (*allkiri*)

Annotatsioon

Käesoleva töö aluseks on mõte, et olemasolevate reeglite kaevandamise algoritmide jõudlus on aja jooksul järjest paranenud ja nende edasiarendamine ei ole esmatähtis. Selle asemel tuleks leida lahendus probleemile, kuidas saada leitud reeglisüsteemidest kvaliteetsemat infot. Töö eesmärgiks on testida ühe olemasoleva andmekaevealgoritmi jõudlust, veendumaks, et olemasolevad algoritmid on tõepoolest piisavalt efektiivsed. Teiseks pakutakse välja ja realiseeritakse algoritm inforikaste reeglisüsteemide leidmiseks ning võrreldakse kahe algoritmi efektiivsust.

Töö tulemusena selgub, et olemasolevad algoritmid on töös käsitletud andmestike jaoks piisavalt efektiivsed. Samuti valmib võrdlus kahe uuritava algoritmi vahel, millest järeldatakse, et inforikaste reeglisüsteemide algoritm töötab sama hea jõudlusega kui olemasolevad algoritmid.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 34 leheküljel, 5 peatükki, 15 joonist, 8 tabelit.

Abstract

This thesis is based on the idea that performance of existing pattern mining algorithms has increased over time and increasing it further is not the main priority. Instead of that, focus should be on improving the quality of the information received from found pattern systems. The aim of the thesis is to test the performance of an already existing algorithm, in order to make sure that the existing algorithms are indeed efficient enough. Secondly, an algorithm is proposed for finding information rich pattern systems and the efficiency of the two algorithms is compared.

As a result of the thesis, it is concluded that the existing algorithms are efficient enough for the datasets used in this work. It is also found that permormance of the algorithm for information rich pattern systems is as good as performance of the existing algorithms.

The thesis is in Estonian and contains 34 pages of text, 5 chapters, 15 figures and 8 tables.

Lühendite ja mõistete sõnastik

FP	***Frequent pattern*** Reegel, mis esineb andmestikus sagedasti
FP-puu	***FP-tree*** Kompaktne struktuur, mis hoiab endas infot andmebaasis sagedasti esinevate reeglite kohta [2]
Plaat	***Tile*** Ala andmestikus, mis määratletakse kõikide objektide alamhulga ja kõikide ridade alamhulga abil [6]
Reegel	***Pattern*** objektide kogum, mis esineb ühes andmebaasi transaktsioonis
Reeglisüsteemide kaevandamine	***Association rule learning*** Meetod leidmaks huvitavaid seoseid andmebaasi muutujate vahel [1]
Tingimuslik FP-puu	***Conditional FP-tree*** FP-puu, mis koosneb ainult transaktsioonidest, mis sisaldavad mingit kindlat objektihulka [3]
Toetus	***Support*** Vaadeldavat objektihulka sisaldavate transaktsioonide osakaal [1]

Jooniste nimekiri

Tabelite nimekiri

Sisukord

1. Sissejuhatus

Andmekaevandamise üks olulisemaid probleeme on reeglite (*patterns*) leidmine. Reeglite leidmiseks on loodud erinevaid algoritme. Algoritmide jõudlus on aja jooksul oluliselt paranenud ja inimtekkeliste andmestike analüüsimine on piisavalt kiire. Samas ei ole suur osa algoritmide poolt leitud reeglitest huvitavad, aga eesmärgiks on leida need, mis on huvitavad. Seega ei ole tänapäeval peamine probleem algoritmide efektiivsuse tõstmine, vaid pigem reeglisüsteemist saadava info kvaliteedi parandamine.

1.1 Taust ja probleem

Reeglisüsteeme kaevandavate algoritmide algusajal oli peamiseks probleemiks algoritmi jõudlus. Tänapäeval on probleemiks pigem reeglisüsteemist saadava info kvaliteet. Algoritm võib leida suhteliselt väikesest andmestikust suure hulga reegleid, aga tekib küsimus, mida sellise tulemusega peale hakata.

See probleem on peaaegu sama vana, kui reeglite otsimine ise. Seni on seda püütud lahendada tehniliselt, proovides leida matemaatilist või heuristilist lahendust. Nende lähenemiste nõrkus on, et nad proovivad samaaegselt olla efektiivsed andmekaevealgoritmid, aga pakkuda ka väga head sisulist reeglisüsteemi. Käesolevas töös uuritakse, kas on mõtet luua reeglite kaevandamiseks ülikiiret algoritmi, kui pärast reeglitest huvitavate välja otsimine vajab ikkagi mahukaid andmeoperatsioone.

1.2 Ülesande püstitus

Lõputöö eesmärgiks on kontrollida hüpoteesi, et inforikaste algoritmide kasutamine reeglite kaevandamiseks on sama efektiivne kui ülimalt optimiseeritud algoritmide kasutamine, mis osa infost reeglite kohta optimiseerimise käigus kaotavad. Soovitakse näidata, et inimtekkeliste andmestike jaoks, mille mõõdud jäävad antud töös kirjeldatud piiridesse, ei ole esmase tähtsusega algoritmi optimiseerimine, vaid algoritmist saadavate tulemuste kvaliteet.

1.3 Metoodika

Püstitatud hüpoteesi kontrollimiseks võrreldakse kahte algoritmi. Üheks võrreldavaks on Christian Borgelti loodud FP-growth algoritm ja teise algoritmina luuakse inforikaste reeglite kaevandamise algoritm. Algoritmide efektiivsuse võrdlemiseks viiakse läbi jõudluse testid, mille abil hinnatakse algoritmide empiirilist keerukust. Selleks, et näidata, millist infot kumbki algoritm säilitab, tuuakse ära mõlema algoritmi tulemusel saadud reeglid.

1.4 Ülevaade tööst

Töö teises peatükis tutvustatakse reeglisüsteemide kaevandamist ja sellega seotud probleeme. Kolmandas peatükis kirjeldatakse FP-Growth algoritmi ja leitakse empiiriliselt FP-Growth algoritmi keerukus. Töö neljas peatükk sisaldab inforikaste reeglisüsteemide leidmise algoritmi kirjeldust ja algoritmi peal läbi viidud keerukuse teste. Samuti on neljandas peatükis algoritmide võrdluse tulemused ja tehtud järeldused.

2. Reeglisüsteemide kaevandamine

Reeglisüsteemide kaevandamine on meetod leidmaks huvitavaid seoseid andmebaasi muutujate vahel. Algselt tutvustasi reeglisüsteemide kaevandamist kui võimalust leida seaduspärasusi toodete vahel kaupluste transaktsioonide andmetes. Näiteks võib meetodi abil leida reegli, et kui inimene ostab samal ajal nii leiba kui saia, siis ostab ta ka piima. Seda informatsiooni võib ära kasutada näiteks toodete paigutamisel. Reeglisüsteeme kasutatakse ka paljudes teistes valdkondades. [1]

2.1 Definitsioon

Reeglisüsteemide kaevandamise probleemi võib defineerida järgnevalt:

Olgu I = {i1, i2, ..., in} binaaratribuutide hulk, mida kutsutakse objektideks.

Olgu D = {t1, t2, ..., tn} transaktsioonide hulk, mida kutsutakse andmebaasiks.

Igal transaktsioonil hulgas D on unikaalne transaktsiooni ID ja iga transaktsioon sisaldab mingit I alamhulka.

Reegel defineeritakse kui implikatsioon:

$X \Rightarrow Y$,

kus $X,Y \subseteq I$ ja $X \cap Y = \emptyset$.

Iga reegel koosneb kahest erinevast objektide hulgast X ja Y, kus X on eeldus ja Y järeldus. [1]

2.2 Mõisted

Selleks, et leida kõikide reeglite hulgast huvitavaid reegleid, kasutatakse erinevaid väärtusi.

Olgu X objektide hulk, $X \Rightarrow Y$ reegel ja T antud andmebaasi transaktsioonide hulk.

Toetus (*support*): X-i toetus T suhtes näitab, kui suur osa andmebaasi transaktsioonidest sisaldab hulka X. Valemi kujul kirjutatakse see järgnevalt: supp(X). [1]

Usaldus (*confidence*): Reegli X ⇒ Y usaldus transaktsioonide T suhtes näitab, kui suur osa transaktsioonidest, mis sisaldavad hulka X, sisaldavad ka hulka Y. Usaldus defineeritakse järgnevalt:

$conf(X \Rightarrow Y) = supp(X \cup Y)/supp(X)$. [1]

Lift defineeritakse järgnevalt:

$lift(X \Rightarrow Y) = \frac{supp(X \cup Y)}{supp(X) \times supp(Y)}$. [1]

Kui reegli *lift* on 1, siis see tähendab, et eelduse ja järelduse esinemissagedus on teineteisest sõltumatud. Kui kaks sündmust on teineteisest sõltumatud, siis ei saa neist tuletada ühtegi reeglit. Kui *lift* on suurem kui 1, siis see näitab, kui suurel määral on kaks sündmust teineteisest sõltuvad. [1]

Veendumus (conviction) defineeritakse järgnevalt:

$conv(X \Rightarrow Y) = 1 - \frac{supp(Y)}{1 - conf(X \Rightarrow Y)}$. [1]

2.3 Protsess

Üldjuhul peab leitavate reeglite toetus ja usaldus olema suurem kui kasutaja poolt määratud miinimum. Reeglite loomine jagatakse tavaliselt kaheks osaks. Kõigepealt leitakse andmebaasist kõik sagedased objektide hulgad, mille toetus on suurem kui minimaalne toetus. Teiseks moodustatakse leitud hulkadest reeglid, nii et reeglite veendumus oleks suurem kui minimaalne veendumus. [1]

Selleks, et leida kõik sagedased objektide hulgad, tuleb otsida üles kõik võimalikud hulgad. Võimalike hulkade arv on $2^n - 1$. See tähendab, et otsitavate hulkade arv kasvab objektide arvu suurenedes eksponentsiaalselt. Efektiivne otsimine on siiski võimalik, kasutades ära toetuse omadust, mille põhjal kõik sagedase objektihulga alamhulgad on samuti sagedasti esinevad. Järelikult kõik mittesagedase hulga ülemhulgad on samuti mittesagedased. [1]

2.4 Algoritmid

Läbi aja on loodud mitmeid erinevaid algoritme reeglisüsteemide leidmiseks. Tuntumad neist on Apriori, Eclat ja FP-Growth. [1]

Apriori on ilmselt kõige tuntum reeglite kaevandamise algoritm. See kasutab laiuti otsingut, et leida objektihulkade toetused. Samuti kasutab see kandidaatide genereerimise funktsiooni. [1]

Eclat on sügavuti otsingu algoritm, mis kasutab hulkade ühisosa. [1]

2.5 Huvitavate reeglite leidmine

Üheks oluliseks küsimuseks reeglite kaevandamisel on kuidas leida kõikide reeglite asemel ainult huvitavad reeglid. Selleks, et otsustada, millised reeglid on huvitavad, kasutatakse erinevaid näitajaid. Huvitavus on oma olemuselt subjektiivne omadus ja seetõttu ei ole ühest huvitavuse näitajat, mis sobiks kõikideks olukordadeks. [6]

2.5.1 Sagedased objektihulgad

Kõige tuntum meetod huvitavate reeglite leidmiseks on sagedaste hulkade kaevandamine, mis tähendab, et leitakse kõik hulgad, mille toetus on suurem kui määratud minimaalne toetus. Idee on siin lihtne: mida rohkem mingi reegel andmete seas esineb, seda huvitavam see on. Tänu toetuse monotoonsusele on sagedaste reeglite leidmine efektiivne. [6]

Sageduse probleem on, et kui määrata kõrge toetuse piirang, siis leitakse ainult reeglid, mis esitavad juba üldlevinud teadmisi. Seevastu piirangut langetades, saadakse tihtipeale väga palju üleliigseid reegleid ja kõigi nende läbi vaatamine võib osutuda võimatuks. Seda probleemi tuntakse reeglite plahvatusena. Probleemi lahendamiseks on proovitud tulemuste hulka vähendada, mis seisneb selles, et leitakse ainult reeglid, millest saab tuletada täieliku reeglite hulga. [6]

Üheks võimalikuks lahenduseks on maksimaalsed sagedased objektihulgad, mis on sagedased objektihulgad, mida ei saa laiendada, ilma et nende toetus langeks alla minimaalse toetuse. Maksimaalsetest sagedastest objektihulkadest saab tuletada kõik sagedased objektihulgad, aga mitte individuaalseid sagedusi. Maksimaalsed objektihulgad on kasulikud, kui meid huvitavad pikad reeglid. [6]

Teine võimalus tulemuste hulka piirata on leida suletud sagedased objektihulgad. Erinevalt maksimaalsetest objektihulkadest saab neist tuletada täieliku sagedaste objektihulkade hulga. Suletud sagedased hulgad defineeritakse kui hulgad, mis on sagedased ja mida ei saa laiendada, ilma et toetus muutuks. Kui on olemas suletud sagedased objektihulgad, siis saab nende abil kergesti määrata teiste objektihulkade toetuse. Objektihulga toetuse leidmiseks, leitakse selle väikseim ülemhulk, mis kuulub suletud sagedaste objektihulkade hulka ja tagastatakse selle toetus. Kui sellist ülemhulka ei leidu, siis ei ole objektihulk sagedane. [6]

Suletud sagedaste objektihulkadega on lähedalt seotud vabad sagedased objektihulgad. Kui suletud objektihulki võib võtta kui maksimaalseid objektihulki, mille toetus on sama, siis vabad hulgad on sama toetusega minimaalsed objektihulgad. [6]

2.5.2 Plaadid

Lisaks sagedastele objektihulkadele võib kaevandada ka plaate. Plaadid on alad andmestikus, mis määratletakse kõikide esemete alamhulga ja kõikide ridade alamhulga abil. [6]

Esimene plaatide kaevandamise meetod on suurte plaatide kaevandamine. Siin leitakse plaadid, mis koosnevad ainult 1-dest ja mis on suuremad või sama suured kui määratud minimaalne suurus. [6]

Kuna ainult 1-sid sisaldavate plaatide leidmine ei pruugi olla realistlik kasutatakse ka mürarikaste plaatide kaevandamist. Mürarikkad plaadid on seotud 1-de sagedusega andmetes. Neid kaevandades proovitakse leida suuri alasid, mis sisaldavad palju ühtesid või nulle. Mida ühtsemad on plaadil olevad väärtused, seda huvitavam on plaat. [6]

3. FP-Growth algoritm

FP-Growth algoritm võimaldab avastada sagedasti esinevaid objektihulki, ilma kandidaatobjektihulki genereerimata. Algoritmi töö võib jagada kaheks osaks:

- 1. samm: ehitada kompaktne andmestruktuur ehk FP-puu;
- 2. samm: eraldada FP-puust sagedased objektihulgad. [3]

3.1 FP-puu struktuur

FP-puu ehk sagedaste reeglite puu on kompaktne struktuur, mis hoiab endas infot andmebaasis sagedasti esinevate reeglite kohta.

FP-puu koosneb ühest juurelemendist, mille märgendiks on „null“, alamprefiksipuudest ja sagedaste objektide päiste tabelist. Iga sõlm alamprefiksipuus koosneb kolmest väljast:

- objekti nimi: näitab, milline objekt vastab antud sõlmele;
- loendur: näitab transaktsioonide arvu, milles objekt esineb, või teede arvu, mis jõuavad antud sõlmeni;
- Sõlmelink: ühendab sõlme järgmise sõlmega;

Iga kirje sagedaste objektide päiste tabelis koosneb kahest väljast:

- Objekti nimi;
- Viide esimesele objekti nimega sõlmele FP-puus. [2]

3.2 FP-puu ehitamine

FP-puu koostamiseks käib algoritm andmestikku kaks korda üle. Esimesel korral uurib ta andmeid ja leiab iga objekti jaoks toetuse (*support*) ehk tema andmebaasis esinemise sageduse. Seejärel jäetakse mittesagedased objektid kõrvale ja järelejäänud objektid sorteeritakse nii, et sagedasemad objektid on eespool.

Teisel läbikäigul ehitab algoritm FP-puu. FP-puu sõlmed on vastavuses andmebaasi objektidega ja igal sõlmel on loendur. Puu ehitamiseks loeb algoritm kordamööda transaktsioone ja lisab need puule. Igas transaktsioonis on objektid sorteeritud sageduse järgi. Nii toimides võivad transaktsioonide teed ristuda, kui transaktsioonidel on samad objektid. Sellistel puhkudel kasutatakse loendurit. Mida rohkem teid ristub, seda väiksem on puu mälumaht ja seda kiirem on puu töötlemine. [3]

3.2.1 Kirjeldus näite abil

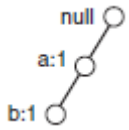
Selleks, et algoritmi tööd täpsemalt kirjeldada, toome ühe näite. Järgnev näide on võetud Pang-Ning Tani, Michael Steinbachi ja Vipin Kumari raamatust „Introduction to Data Mining“ [5]. Olgu meil andmestik, kus on 10 transaktsiooni ja 5 objekti. Andmestik on kujutatud tabelis 1.

Tabel 1. Andmestik transaktsioonidega. [5]

TID	Objektid
1	{a,b}
2	{b,c,d}
3	{a,c,d,e}
4	{a,d,e}
5	{a,b,c}
6	{a,b,c,d}
7	{a}
8	{a,b,c}
9	{a,b,d}
10	{b,c,e}

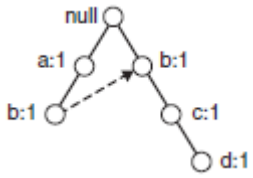
Alguses on FP-puus ainult juursõlm, mille märgendiks on *null*. Esimese sammuna käib algoritm kõik andmed üle. Mittesagedased objektid jäetakse kõrvale, sagedased objektid sorteeritakse toetuse järgi. Antud näites on kõige sagedasem objekt *a*, talle järgnevad *b*, *c*, *d* ja *e*.

Selleks, et ehitada FP-puu käib algoritm teist korda andmed läbi. Kõigepealt loeb ta esimest transaktsiooni {*a*,*b*} ja loob sõlmed *a* ja *b*. Seejärel moodustatakse tee *null* → *a* → *b* ja iga tee peal oleva sõlme loendur saab väärtuseks 1. Saadud puu on kujutatud joonisel 1.



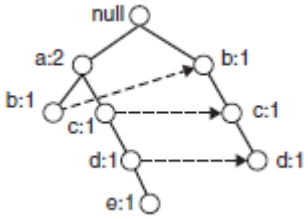
Joonis 1. FP-puu pärast esimese transaktsiooni lugemist. [5]

Pärast teise transaktsiooni {*b*,*c*,*d*} lugemist luuakse uued sõlmed *b*, *c* ja *d* ja moodustatakse uus tee *null* → *b* → *c* → *d*. Iga sõlme loenduri väärtus on 1. Kuigi kahel esimesel transaktsioonil on sama objekt *b*, on nende teed eraldi, sest transaktsioonidel ei ole sama prefiksi. Joonisel 2 on kujutatud puu pärast teise transaktsiooni lugemist.



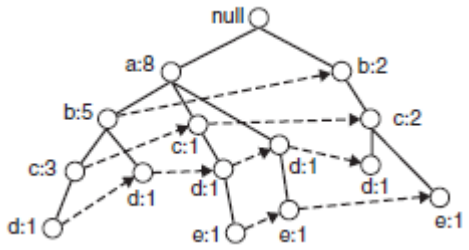
Joonis 2. FP-puu pärast teise transaktsiooni lugemist. [5]

Kolmandal transaktsioonil on sama prefiks, mis esimesel transaktsioonil, milleks on objekt *a*. Selle tõttu kattub kolmanda transaktsiooni tee *null* → *a* → *c* → *d* → *e* osaliselt esimese transaktsiooni teega. Saadud tee on nähtav joonisel 3. Kuna transaktsioonide teed kattuvad, siis suurendatakse sõlme *a* loendurit ühe võrra.



Joonis 3. FP-puu pärast kolmanda transaktsiooni lugemist. [5]

Sarnast protsessi jätkatakse kuni iga transaktsioon on lisatud FP-puule. Lõplik FP-puu on kujutatud joonisel 4.



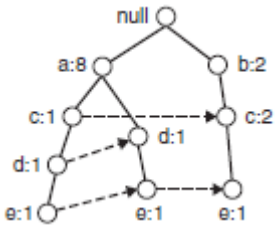
Joonis 4. FP-puu pärast kõikide transaktsioonide lugemist. [5]

3.3 Sagedaste objektihulkade genereerimine

FP-growth leiab FP-puust sagedased objektihulgad uurides puud altpoolt ülespoole. Eespool toodud näites proovib algoritm kõigepealt leida objektihulki, mis lõppevad *e*-ga, seejärel *d*, *c*, *b* ja *a*-ga lõppevaid hulki. Selleks, et saada kindla objektiga lõppevaid objektihulki, on vaja uurida ainult neid teid, mis sisaldavad antud objekti.

Teatud kindla objektiga lõppevate objektihulkade leidmiseks kasutab algoritm „Jaga ja valitse“ meetodit, mis jagab probleemi väiksemateks alamprobleemideks. Oletame, et me tahame leida *e*-ga lõppevaid sagedasi hulki. Alustuseks, peame me kontrollima, kas objektihulk {*e*} ise on sagedane. Kui on, siis proovitakse leida objektihulki, mis lõppevad *de*-ga, *ce*-ga, *be*-ga, *ae*-ga. Need alamprobleemid jagatakse omakorda väiksemateks alamprobleemideks. Liites kõikide alamprobleemide lahendused, saame me kõik sagedased objektihulgad.

Selleks, et kirjeldada konkreetse alamprobleemi lahendamist, uurime *e*-ga lõppevate objektihulkade leidmist. Esimese sammuna kogutakse kokku teed, mis sisaldavad *e*-d. Neid teid kutsutakse prefiksiteedeks ja need on kujutatud joonisel 5.



Joonis 5. Prefiksiteed, mis lõppevad *e*-ga. [5]

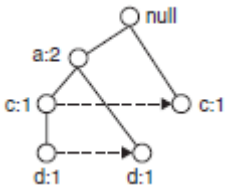
Prefiksiteedelt leitakse *e* toetus liites kokku kõikide *e*-sõlmede toetused. Saame, et *e* toetus on 3. Eeldusel, et minimaalne toetus on 2, on $\{e\}$ sagedane objektihulk.

Kuna $\{e\}$ on sagedane, on vaja leida *de*, *ce*, *be* ja *ae*-ga lõppevad sagedased objektihulgad. Enne nende alamprobleemide lahendamist, peame me teisendama prefiksiteed tingimuslikuks FP-puuks. See on struktuurselt sarnane FP-puule, aga see on mõeldud kindla sufiksiga alamhulkade leidmiseks.

Selleks, et saada tingimuslikku FP-puud, on vaja kõigepealt värskendada prefiksiteedel olevaid toetuse loendureid. Tuleb eemaldada toetus transaktsioonidelt, mis ei sisalda *e*-d. Näiteks tee *null* → *b:2* → *c:2* → *e:1* sisaldab transaktsiooni $\{b, c\}$, milles ei ole objekti *e*. Seega tuleb kõik teel olevad toetused langetada 1-le.

Seejärel eemaldadatakse *e* sõlmed, sest toetuse loendurid on värskendatud ja alamprobleemide lahendamiseks ei ole vaja informatsiooni *e* kohta. Samuti eemaldatakse objektid, mis ei ole enam sagedased. Näiteks sõlm *b* esineb ainult üks kord ja selle toetus on 1. Seega võib objekti *b* jätta edasisest analüüsist kõrvale, sest kõik objektihulgad, mis lõppevad *be*-ga on mittesagedased.

Joonisel 6 on tingimuslik FP-puu. Võrreldes prefiksiteedega on sellel puudu sõlmed *b* ja *e* ning sagedusloendurid on värskendatud.



Joonis 6. Tingimuslik FP-puu *e* jaoks. [5]

Järgmisena tuleb lahendada eespool kirjeldatud alamprobleemid. Selleks, et leida *de*-ga lõppevad objektihulgad, kogutakse tingimuslikust FP-puust kokku prefiksiteed *e* jaoks. Liites sõlme *d*-ga seotud toetused, saadakse hulga {*d,e*} toetus. Kuna toetus on 2, on {*d,e*} sagedane objektihulk. Järgmisena moodustatakse tingimuslik FP-puu *de* jaoks. Pärast toetuste värskendamist ja mittesagedase objekti *c* eemaldamist saadud tingimuslik FP-puu on näha joonisel 7.

Joonis 7. Tingimuslik FP-puu *de* jaoks. [5]

Sellest tingimuslikust FP-puust leiab algoritm sagedase objektihulga {*a,d,e*}. Kuna siin rohkem objekte ei ole, liigub algoritm edasi järgmise alamprobleemi juurde, milleks on *ce*-ga lõppevate sagedaste objektihulkade leidmine. Sealt leiab ta objektihulga {*c,e*} ja järgmist alamprobleemi lahendades leiab ta hulga {*a,e*}.

Olles leidnud kõik *e*-ga lõppevad sagedased objektihulgad, leiab algoritm samamoodi ka ülejäänud sufiksitega sagedased objektihulgad. Kõikidelt teedelt leitud sagedased hulgad on toodud välja tabelis 2.

Tabel 2. Kõikidele sufiksitele vastavad sagedased objektihulgad. [5]

Sufiks	Sagedased objektihulgad
e	{e}, {d,e}, {a,d,e}, {c,e}, {a,e}
d	{d}, {c,d}, {b,c,d}, {a,c,d}, {b,d}, {a,b,d}, {a,d}
c	{c}, {b,c}, {a,b,c}, {a,c}

b	{b}, {a,b}
a	{a}

3.4 Keerukuse empiiriline leidmine

Selleks, et analüüsida algoritmi keerukust on kaks võimalust. Üks võimalus on arvutada keerukus teoreetiliselt, analüüsides algoritmi tööpõhimõtet. Teine võimalus, mis arvestab ka realisatsiooni ja andmestiku eripäradega, on testida algoritmi jõudlust empiiriliselt katsete käigus. Selles töös testitakse algoritmi töö aega erinevate sisendandmetega. Uuritakse algoritmi ajalist keerukust sõltuvalt sisendandmestiku ridade arvust, veergude arvust ja erinevate objektide arvust.

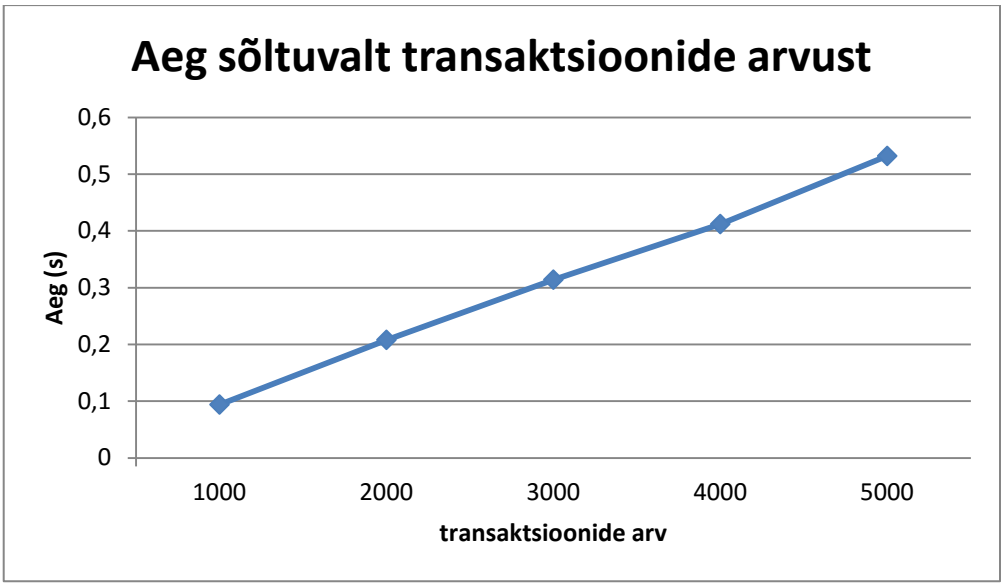
FP-growth keerukuse hindamiseks kasutatakse Christian Borgelti C keeles kirjutatud realisatsiooni [4], mis on üks kõige efektiivsemaid FP-growthi realisatsioone. Testimiseks genereeritakse andmestikud, milles objektideks on juhuslikud arvud. Iga erineva suurusega andmestiku töötlemiseks kulunud aja leidmiseks tehakse 10 katset ja leitakse nende keskmine.

3.4.1 Ajaline keerukus sõltuvalt transaktsioonide arvust

Testitakse algoritmi töö aega viie erineva transaktsioonide arvuga andmestiku korral. Kõikides andmestikes on transaktsioonis 70 objekti ja erinevate objektide arv on 100. Saadud tulemused on toodud tabelis 3 ja joonisel 8.

Tabel 3. Algoritmi töö aeg sõltuvalt transaktsioonide arvust.

Transaktsioonide arv	1000	2000	3000	4000	5000
Keskmine aeg (s)	0,094	0,21	0,31	0,41	0,53



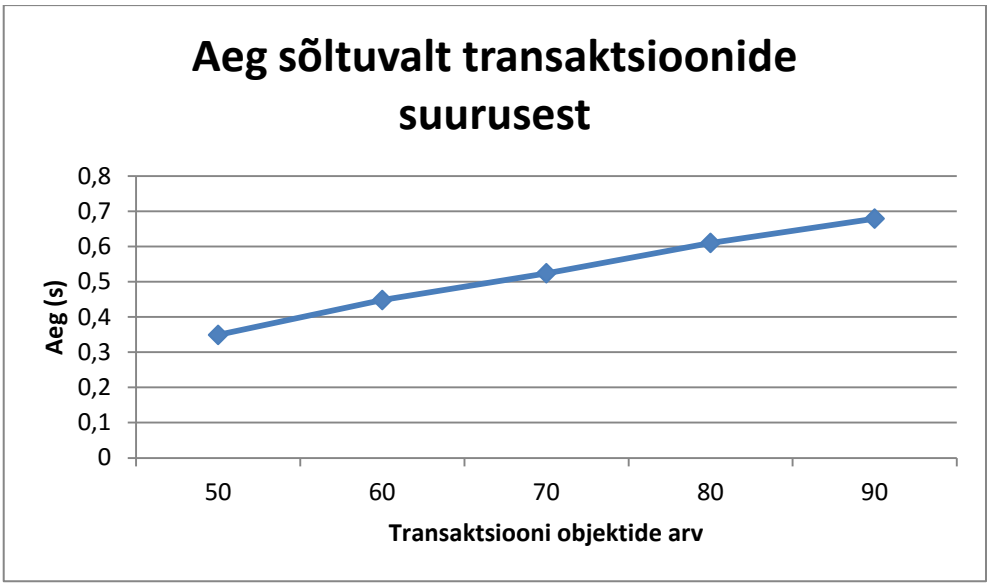
Joonis 8. Algoritmi töö aeg sõltuvalt transaktsioonide arvust.

3.4.2 Ajaline keerukus sõltuvalt objektide arvust transaktsioonis

Leitakse algoritmi töö aeg viie erineva transaktsioonide suuruse korral. Igas testitavas andmestikus on 5000 transaktsiooni ja 100 erinevat objekti. Testide tulemused on toodud tabelis viis ja joonisel 9.

Tabel 4. Algoritmi töö aeg sõltuvalt transaktsioonide suurusest.

Transaktsioonide suurus	50	60	70	80	90
Keskmine aeg (s)	0,35	0,45	0,52	0,61	0,68



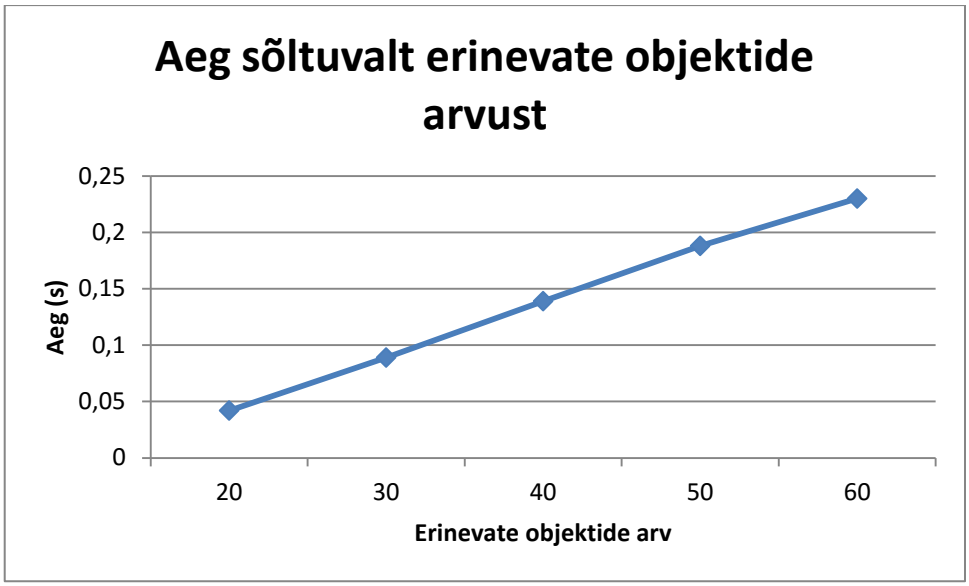
Joonis 9. Algoritmi töö aeg sõltuvalt transaktsioonide suurusest.

3.4.3 Ajaline keerukus sõltuvalt erinevate objektide arvust

Leitakse algoritmi keerukus sõltuvalt erinevate objektide arvust. Igas andmestikus on 5000 transaktsiooni ja igas transaktsioonis on 50 objekti. Saadud tulemused on toodud tabelis 5 ja kujutatud graafiliselt joonisel 10.

Tabel 5. Algoritmi töö aeg sõltuvalt erinevate objektide arvust

Erinevate objektide arv	20	30	40	50	60
Keskmine aeg (s)	0,042	0,089	0,14	0,19	0,23



Joonis 10. Algoritmi töö aeg sõltuvalt erinevate objektide arvust.

4. Inforikkama reeglisüsteemi koostamine

Eelpool kirjeldatud FP-growth algoritm on väga kiire, aga annab tulemusena suure hulga reegleid, millest huvitavate leidmine nõuab omakorda mahukaid andmeoperatsioone. Pakkumaks alternatiivset lahendust, luuakse inforikaste reeglite kaevandamise algoritm, mis säilitab süsteemi kohta rohkem infot kui FP-growth algoritm ja seega teeb huvitavate reeglite leidmise lihtsamaks. Algoritm leiab andmestikust reeglid maatriksarvutuste abil, kasutades selleks standardseid lineaaralgebra teeke.

Reeglite otsimise esimese sammuna leiab algoritm kõikide andmestiku objektide ja objektipaaride sagedused. Selleks korrutab ta etteantud binaarmaatriksi selle transponeeritud maatriksiga. Saadud maatriksi diagonaalilt loetakse üksikute elementide sagedused, maatriksi ülejäänud elemendid esitavad objektipaaride sagedusi. Nende andmete põhjal koostab algoritm neljakihilise tensormaatriksi, kus on info objektipaaride kohta. Maatriksi esimesel kihil on sagedused, kus mõlemad objektid on olemas, teisel kihil on sagedused, kus esimene objekt on ja teist ei ole, kolmandal kihil vastupidi ja neljandal kihil on sagedused, kus kumbagi objekti ei ole. Seejärel valib algoritm reeglid, mille sagedus on kõrgem kui minimaalne sagedus.

4.1 Keerukus

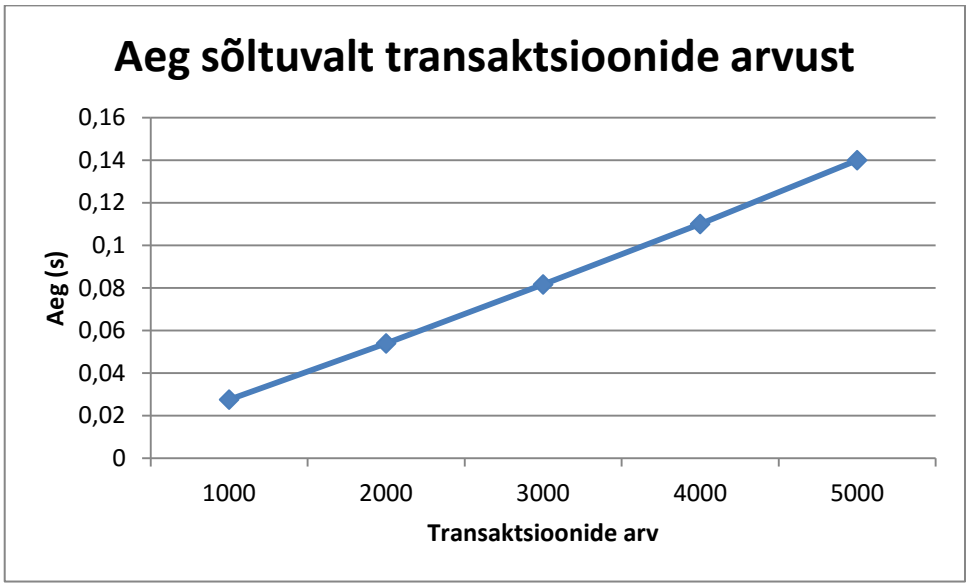
Sarnaselt FP-Growth algoritmile leitakse ka selle algoritmi empiiriline keerukus. Algoritmi keerukuse mõõtmiseks genereeritakse binaarandmestikud. Andmestikud on samasuguste transaktsioonide arvu, transaktsioonide suurusega ja erinevate objektide arvuga nagu FP-Growth testimiseks kasutatavad andmestikud. Iga erineva andmestikuga sooritatakse 10 mõõtmist ja leitakse tulemuste keskmine.

4.1.1 Keerukus sõltuvalt transaktsioonide arvust

Leitakse algoritmi keerukus sõltuvalt transaktsioonide arvust. Kõikidel andmestikel on 100 erinevat objekti ja transaktsioonide suurus on 70. Saadud tulemused on toodud tabelis 6 ja joonisel 11.

Tabel 6. Aeg sõltuvalt transaktsioonide arvust.

Transaktsioonide arv	1000	2000	3000	4000	5000
Keskmine aeg (s)	0,028	0,054	0,082	0,11	0,14



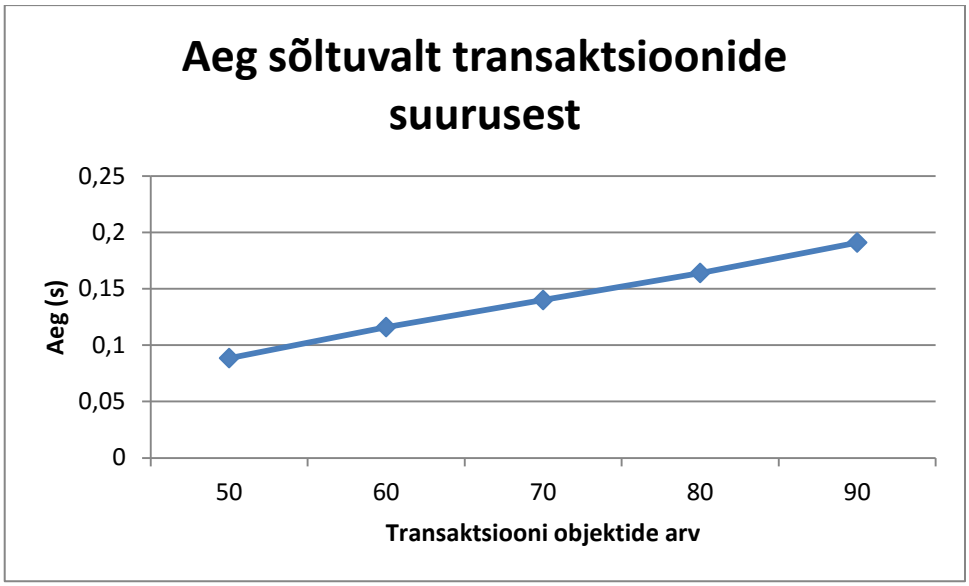
Joonis 11. Aeg sõltuvalt transaktsioonide arvust.

4.1.2 Keerukus sõltuvalt transaktsioonide suurusest

Leitakse algoritmi töö aeg sõltuvalt transaktsioonide suurusest. Igas testitavas andmestikus on 5000 transaktsiooni ja 100 erinevat objekti. Testide tulemused on toodud tabelis 7 ja joonisel 12.

Tabel 7. Aeg sõltuvalt transaktsioonide suurusest.

Transaktsioonide suurus	50	60	70	80	90
Keskmine aeg (s)	0,089	0,12	0,14	0,16	0,19



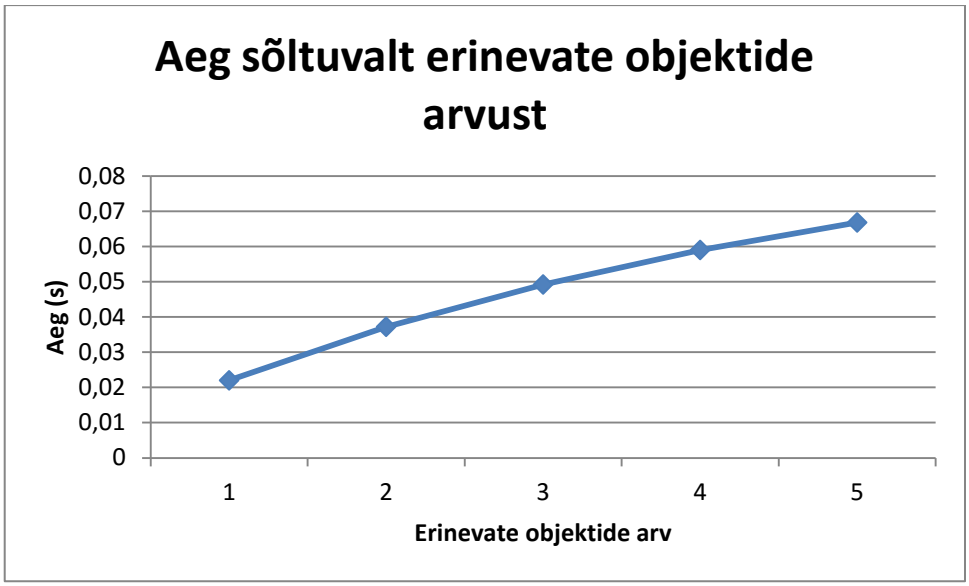
Joonis 12. Aeg sõltuvalt transaktsioonide suurusest.

4.1.3 Keerukus sõltuvalt erinevate objektide arvust

Leitakse algoritmi keerukus sõltuvalt erinevate objektide arvust. Igas andmestikus on 5000 transaktsiooni ja igas transaktsioonis on 50 objekti. Saadud tulemused on toodud tabelis 8 ja kujutatud graafiliselt joonisel 13.

Tabel 8. Aeg sõltuvalt erinevate objektide arvust.

Erinevate objektide arv	20	30	40	50	60
Keskmine aeg (s)	0,022	0,037	0,049	0,059	0,067



Joonis 13. Aeg sõltuvalt erinevate objektide arvust.

4.2 Võrdlus FP-growth algoritmiga

Sooritatud testide tulemusena selgub, et koostatud inforikaste reeglite kaevandamise algoritm töötab kiiremini kui FP-growth algoritm. Keskmiselt kulus inforikkal algoritmil andmestikust reeglite leidmiseks 3,4 korda vähem aega kui FP-growth'il. Siinkohal tuleks arvesse võtta, et FP-growth algoritmi töö aeg sisaldab ka leitud reeglite väljundisse kirjutamise aega, inforikaste süsteemide algoritm arvestab ainult reeglite otsimisele kulunud aega. Seda sellel põhjusel, et FP-growth kirjutab iga reegli juba selle leidmise ajal ja see teeb kirjutamise aja eraldamise reeglite otsimise ajast keeruliseks.

Sellegipoolest võib väita, et inforikaste reeglite kaevandamise algoritm ei jää jõudluse poolest FP-growth algoritmile alla, vaid pigem jõuab tulemuseni kiiremini. Saadud testide tulemused näitavad, et mõlemad algoritmid on väga efektiivsed ja andmestike, mille suurus on sarnane käesolevas töös käsitletud andmestikega, töötlemine ei ole aeganõudev. Kõige suuremal andmestikul, mis antud töös algoritmidele ette anti, on 5000 transaktsiooni ja 90 erinevat objekti. Fp-growth algoritmil kulus antud andmestikust reeglite leidmiseks keskmiselt 0,68 sekundit, inforikaste süsteemide algoritmil kulus selleks 0,19 sekundit.

Järelikult ei ole väiksemate inimtekkeliste andmestike puhul niivõrd oluline algoritmide jõudluse parandamine, pigem tuleks rõhku panna algoritmist saadava info kvaliteedi tõstmisele. Selleks, et võrrelda kummagi algoritmi poolt säilitatavat infot, on toodud näide

reeglist mõlema algoritmi tulemist. FP-growth algoritm tagastab reeglid joonisel 14 näidatud kujul:

```
4 53 (19.1)
```

Joonis 14. FP-growth algoritmi tagastatava reegli näide.

FP-growth algoritmi tagastatud reegel sisaldab objekte 4 ja 53 ning objektipaari protsentuaalset transaktsioonides esinemise sagedust. Inforikaste süsteemide algoritmi poolt leitud reegli näide on esitatud joonisel 15.

```
ans = Pattern 4 & 53:
ans =

ans(:,:,1,1) =  191
ans(:,:,1,2) =  239
ans(:,:,1,3) =  252
ans(:,:,1,4) =  318
```

Joonis 15. Inforikaste reeglisüsteemide algoritmi tagastatava reegli näide.

Siin leiab algoritm sama objektipaari, aga lisaks objektipaari sagedusele on olemas ka lisainfo. Tulemusest saame lugeda, et transaktsioone, kus on mõlemad objektid, on 191. Transaktsioone, kus on ainult objekt 4, aga objekti 53 ei ole, on 239. Transaktsioone, kus on ainult objekt 53, aga objekti 4 ei ole, on 252 ja transaktsioone, kus pole kumbagi objekti, on 318.

Tagastatud tulemustest sõltub, mida saab nendega edasi teha. Inforikaste reeglite algoritmi väljund võimaldab hinnata reegli olulisust lähtuvalt selle komponentidest. FP-growth algoritmi korral saab andmeanalüütik küll võrrelda saadud reegli sagedust andmeridade arvuga, aga ei saa hinnata, kui oluline on see sagedus arvestades reegli objektide ülejäänud sageduste jaotumist andmestikus. Põhimõtteliselt oleks sama info leidmine võimalik ka FP-growth korral, ent takistuseks saab optimeerimistehnika, mis jätab osa infot töötlemisprotsessi jooksul kõrvale. Inforikaste reeglite leidmise algoritm aga jätab optimeerimise spetsiaalsetele lineaaralgebra teekidele, mistõttu on võimalik kompaktselt talletada ja kiirelt ilma andmestikku uuesti läbimata kasutada kogu eeltoodud infot. Võimalik, et kui algoritmide keerukust teoreetiliselt hinnata, oleks FP-growth algoritm efektiivsem, kuid tänu efektiivsele realisatsioonile saavutab inforikaste reeglisüsteemide algoritm praktikas paremad tulemused nii kvalitatiivselt (info reeglite kohta) kui kvantitatiivselt (algoritmi töökiirus).

5. Kokkuvõte

Töös püstitati hüpotees, et reeglite kaevandamiseks inimtekkelistest andmestikest ei ole vaja juba olemasolevate algoritmide jõudlust parandada, sest need on juba piisavalt kiired. Samuti oli eesmärgiks uurida, kas inforikaste reeglite kaevandamise algoritm on sama efektiivne kui täielikult optimiseeritud algoritmid.

Hüpoteesi kontrollimiseks viidi läbi jõudluse empiirilised testid kahe erineva algoritmi jaoks. Selleks koostati inforikaste reeglite kaevandamise algoritm ja võrdluseks võeti FP-growth algoritm. Algoritmide jõudluse uurimiseks anti neile ette erineva suurusega andmestikke ja leiti andmestikest reeglite leidmise aeg.

Töö tulemusena leiti, et mõlemad algoritmid on efektiivsed. Ühegi katses kasutatud andmestiku läbi töötamiseks ei kulunud üle ühe sekundi. Lisaks näitasid katsete tulemused, et inforikaste süsteemide algoritm on keskmiselt 3,4 korda kiirem kui FP-growth algoritm.

Saadud tulemustest võib järeldada, et olemasolevad algoritmid on tõepoolest väga efektiivsed ja inimeste loodud andmestikest reeglite leidmine ei ole nende jaoks probleem. Kinnitust leidis ka hüpotees, et inforikaste reeglite kaevandamise algoritm töötab sama hea efektiivsusega kui tõhusaimad andmekaevealgoritmid. Selle põhjal võib öelda, et tänasel päeval pole peamine prioriteet kiiremate algoritmide loomine, vaid reeglisüsteemide info kvaliteedi parandamine.

Summary

In this thesis, a hypothesis was proposed, that for mining patterns from man-made datasets, it is not necessary to improve the performance of existing algorithms as they are already efficient enough. The purpose was also, to find out if an algorithm for mining information rich pattern systems is as efficient as highly optimized algorithms.

In order to test the hypothesis, empirical complexity tests for two algorithms were carried out. For this, an algorithm for mining information rich patterns was created and compared with FP-growth algorithm. To study performance of the algorithms, algorithms were given datasets of different sizes as an input, and the time it took to find patterns from data was found.

As a result of the thesis, it was concluded that both algorithms are efficient. Processing the datasets used in the tests took no more than 1 second, for any dataset. In addition to this, the tests showed that the algorithm for mining information rich patterns was 3,4 seconds faster than the FP-growth algorithm.

From these results, it can be concluded that existing algorithms are indeed very efficient and finding patterns from man-made datasets is not a problem. Also, the hypothesis, that algorithms for mining information rich patterns are as effective as the most efficient algorithms, was confirmed. On this basis, it can be said, that today the main priority is not making faster algorithms, but improving the quality of found pattern systems.

Kasutatud kirjandus

1. Association rule learning. [WWW] https://en.wikipedia.org/wiki/Association_rule_learning (16.04.2016)
2. Data Mining Algorithms In R/Frequent Pattern Mining/The FP-Growth Algorithm. [WWW] https://en.wikibooks.org/wiki/Data_Mining_Algorithms_In_R/Frequent_Pattern_Mining/The_FP-Growth_Algorithm (23.04.2016)
3. Florian Verhein. Frequent Pattern Growth (FP-Growth) Algorithm: An Introduction, 2008. [WWW] http://wimleers.com/sites/wimleers.com/files/FP-Growth%20presentation%20handouts%20%E2%80%94%C2%A0Florian%20Verhein.pdf (5.04.2016)
4. FP-Growth algorithm – Christian Borgelt's Web Pages. [WWW] http://www.borgelt.net/fpgrowth.html (6.05.2016)
5. Tan, P.-N., Steinbach, M., Kumar, V. Introduction to Data Mining: Chapter 6. Association Analysis: Basic Concepts and Algorithms. [WWW] http://www-users.cs.umn.edu/~kumar/dmbook/ch6.pdf (26.04.2016)
6. Vreeken, J., Tatti, N. Interesting Patterns. In: Aggarwal, C.C., Han, J (eds). Frequent Pattern Mining. Springer, 2014. [WWW] http://eda.mmci.uni-saarland.de/pubs/2014/fpmbook_int-vreeken,tatti.pdf (20.05.2016)

Lisa 1

Octave’s tehtud inforikaste reeglite kaevandamise algoritmi kood.

```
# load sparse dataset
load("random_gen.txt");
dataset=binary;
size(dataset);

minsupport = 10;

tic # start timer

sizeOfData = size(dataset);
numberOfRows = sizeOfData(1);
numberOfCols = sizeOfData(2);
minsupportFreq = numberOfRows/100 * minsupport; #frequency required to meet
minimum support

# find frequencies
freq = full(dataset'*dataset);

diagonal = full(diag(freq));
transposeDiagonal = full(diag(freq)');

#["1:1, 1:0, 0:1, 0:0"]
# result is a 4D tensor matrix, where information is divided by dimensions:
# 1. dimension: frequencies, where both items are 1
# 2. dimension: frequencies, where first item is 1 and other is 0
# 3. dimension: frequencies, where first item is 0 and other is 1
# 4. dimension: frequencies, where both items are 0
result = cat(4, freq, (diagonal-freq), (transposeDiagonal-freq),
(numberOfRows.+freq-diagonal-transposeDiagonal));

resultIndex = freq(:,:) >= minsupportFreq; #select all results with minsupport

resultIndex = triu(resultIndex, 1); #select only rules above the diagonal (it is
symmetrical matrix and below are duplicates)

# save row indexes of selected elements in "row" and column indexes in "col"
[row, col] = find(resultIndex);

toc # stop timer
```