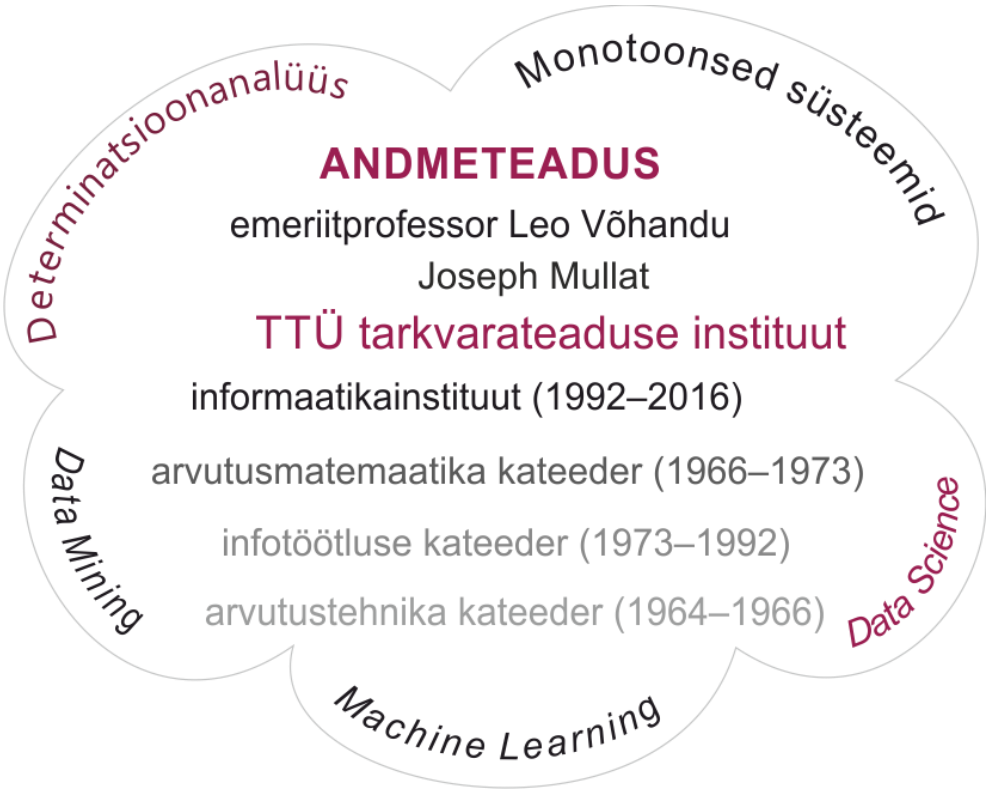
TALLINNA
TEHNIKAÜLIKOOL

Monotoonsed süsteemid ja nende rakendused

ISBN 978-9949-83-462-4

Leo Võhandu, Rein Kuusik, Grete Lind



Tallinn 2018

SISUKORD

EESSÕNA

Käesolev raamat edastab pisikest osa teadustööst, mida andmeteaduse (*Data Science*) valdkonnas on enam kui 50 aasta jooksul emeriitprofessor Leo Võhandu juhtimisel tehtud TTÜ tarkvarateaduse instituudis, mille ühtedeks eellasteks olid informaatikainstituut (1992–2016), infotöötluse kateeder (1973–1992), arvutusmatemaatika kateeder (1966–1973) ja arvutustehnika kateeder (1964–1966).

Professor Leo Võhandu ja dotsent J. Mullat arendasid 1970-ndatel aastatel uue monotoonsete süsteemide teooria, mis võimaldas luua kiireid algoritme andmetöötluse tarbeks. J. Mullat lõi käsitluse, mille esitas oma doktoritöös 1973. a ja publitseeris artiklitena 1971-1977 (Mullat 1971, 1976, 1977a, 1977b, Mullat & Võhandu 1979). Monotoonsete süsteemide nime võttis kasutusele J. Mullat 1976.a. (Mullat 1976). J. Mullati hilisemad artiklid MS teooria valdkonnas on kätte saadavad TTÜ digikogus (https://ws.lib.ttu.ee/digibase/Publ/Search). Leo Võhandu monotoonsete süsteemide rakendused on publitseeritud artiklites (*Выханду* 1979, 1980, 1981, Vyhandu 1989, Võhandu et al. 2006).

Kõik käesolevas raamatus esitatud algoritmid ja meetodid rakendavad vähemal või suuremal määral nimetatud teooriat. Kuna meetodite kasutajaskonnaks olid valdavalt avaliku arvamuse ja turu-uuringu firmad (ankeetküsitlused), siis kogu loodud metoodika ja algoritmika lähtus andmetabelist, mitte aga andmebaasist – töödeldavad andmed anti ette struktuurse objekt-tunnus tabelina.

Raamatus käsitletakse lähemalt monotoonsete süsteemide teoorial baseeruvaid meetodeid ja algoritmikat andmekaeve ja masinõppe valdkonnas. Andmetabeli **korrastusmeetodid** ning nn **Mullati tuumad** on loodud juba 1970-ndatel aastatel, **Hüpoteeside generaator** ja mitmed **determinatsioon-analüüsi (DA) meetodid** loodi algselt 1980–1990-ndatel aastatel, uued arendused 2000-ndatel, nn **nullfaktori vaba DA** aga peale 2010-ndat aastat.

Kuna mitmed käsitlused ja baasalgoritmid on loodud enne Eesti taasiseseisvumist, siis käesolevas töös esitatud meetodid ja algoritmika on vähe mõjutatud valdkonna maailma parimate teadurite poolt – arendusideed on pigem kujunenud oma koolkonna keskselt.

Andmetabelite korrastamise teemat on väga põhjalikult käsitlenud meie kolleeg Tallinna Tehnikaülikoolist professor Innar Liiv oma doktoritöös „Mustrite kasutamine kasutades järjestamist ning maatriksi ümberkorrastamist: unifitseeritud vaade, edasiarendused ning rakendus ladude juhtimises" (Pattern Discovery Using Seriation and Matrix Reordering: A Unified View, Extensions and an Application to Inventory Management). See töö tunnistati organisatsiooni *Classification Society* poolt selle valdkonna parimaks doktoritööks 2008. a (https://tcs.wildapricot.org/Dissertation-Award).

Kõikide raamatus kirjeldatud algoritmide tööd selgitatakse tekstis näiteandmetabelite peal, muutmaks nähtavaks olulisemaid nüansse nende töös. YouTube'is on ka algoritmide samm-sammulist tööd demonstreerivad videod: konformismiskaala, mõjuskaala, plusstehnika, miinustehnika, segatehnika, Mullati tuumad, kõik maksimaalsed klikid (KMK), Pardalose algoritm, otsustuspuu, hüpoteeside generaator, MONSIL, determinatsioonanalüüs.

Paljude antud raamatus käsitletud rakenduste tarbeks on olemas mitmeid teisi meetodeid teistelt autoritelt (R. Agrawal, M. J. Zaki, T. Uno, H. Mannila jt). Neist igaühel on oma algoritmika ja spetsiifilised nõuded algandmete esitusele, oma tugevused ja nõrkused. Vaieldamatult kiireim nendest meetoditest on Takeaki Uno LCM-meetod suletud hulkade leidmiseks, mis baseerub sagedaste hulkade leidmise erilisel, väga efektiivsel ja ratsionaalsel tehnikal. Seevastu monotoonsete süsteemide teooria on üldine teooria, mida lisaks andmeteadusele oleme rakendanud edukalt ka teistes ainevaldkondades, nagu näiteks otsustuspuude formeerimisel, kõikide maksimaalsete klikkide ja suurima kliki leidmisel, disjunktiivsete normaalkujude minimeerimisel jms. Osa nende rakenduste tulemustest on publitseeritud teadusartiklitena, osa seevastu on publitseerimata, kuid on osaliselt kättesaadavad TTÜ digiraamatukogus doktori- või magistritöödena. Raamatu autorite poolt juhendatud andmeteaduse valdkonna lõputööde osaline loetelu on toodud raamatu lisas.

Lisaks monotoonsete süsteemide teooria rakenduste huvilistele on see raamat sobilik ka neile, kes õpivad andmestruktuure. Nimelt mitmete kirjeldatud algoritmide efektiivne realisatsioon eeldab sobivate andmestruktuuride kasutamist – erinevad andmestruktuurid mõjutavad oluliselt realisatsiooni töökiirust. Seetõttu raamatus käsitletud algoritmid on heaks proovikiviks andmestruktuuridega katsetamisel.

I MONOTOONSED SÜSTEEMID

Sissejuhatus

Sagedaseks ülesandeks keeruliste süsteemide käitumise uurimisel on konkreetse süsteemi arvandmete (nt süsteemi kirjeldav N*M andmetabel, kus N on tabeli ridade, M aga veergude arv) analüüs. Andmete alusel tuleb selgitada, kas süsteemis esinevad erilised elemendid või elementide rühmad (allsüsteemid), mis reageerivad mingitele mõjutustele ühtemoodi, samuti selliste allsüsteemide omavahelisi suhteid. Teisiti öeldes, tuleb leida süsteemi struktuur.

Süsteemi struktuur on süsteemi elementide selline organiseerumine allsüsteemideks, mis väljendub allsüsteemide vaheliste suhete hulgana. Süsteemi struktuuriks võib olla näiteks meetod allsüsteemide ühendamiseks terviklikuks süsteemiks, kui see toimub süsteemi elementide vaheliste tugevate ja nõrkade seoste alusel.

Süsteemiteoorias tavaliselt vaadeldakse mittevahetuid seoseid elementide vahel. Sellised seosed on dünaamilised selles mõttes, et seose tase määratakse allsüsteemi poolt, milles seda seost vaadeldakse. Allpool käsitletaksegi ühte sellist dünaamiliste süsteemide klassi – monotoonseid süsteeme (Mullat 1971, 1976, 1977a, 1977b, *Выханду* 1980).

Monotoonsete süsteemide omadused võimaldavad üldisel kujul formuleerida süsteemi tuuma kui allsüsteemi, mis peegeldab kogu süsteemi struktuuri tervikuna. Tuum on allsüsteem, mille elemendid on tundlikud kahe tegevuse (pluss või miinus (Mullat 1976)) suhtes, kusjuures see tundlikkus tegevuste suhtes on määratud süsteemi sisemise struktuuriga. Pluss- ja miinustegevuste sissetoomine põhjustab kahte liiki tuumade olemasolu – pluss- ja miinustuumad.

Tuumade (eriliste omadustega allsüsteemide) olemasolu garanteeritakse vastava matemaatilise mudeliga. Tuumade eraldamine kujutab aga endast tüüpilist suure süsteemi väikeste süsteemide (tuumade) kaudu kirjeldamise ülesannet. Selles mõttes tuum kujutab endast allsüsteemi, mille eemaldamine süsteemist kardinaalselt muudab selle omadusi: süsteem kaotab oma esialgse struktuuri.

Põhimõisted

Defineerime põhimõisted lähtuvalt Mullati töödest (1971, 1976).

Definitsioon 1. Olgu antud lõplik hulk X, |X|=K, ja sellel funktsioon π_X, mis seab igale elemendile $a \in X$ vastavusse väärtuse π_X, $\pi_X(a)$. Funktsiooni π_X nimetatakse **kaalufunktsiooniks**, kui ta on määratud suvalisel alamhulgal $X' \subseteq X$. Väärtust $\pi_X(a)$ nimetatakse elemendi $a \in X$ **kaaluks** hulgal X'.

Definitsioon 2. Hulka X koos kaalufunktsiooniga π_X nimetatakse **süsteemiks** ja tähistatakse $P = (X,\pi_X)$.

Definitsioon 3. Süsteemi $P' = (X',\pi_{X'})$, kus $X' \subseteq X$, nimetatakse süsteemi $P = (X,\pi_X)$ **allsüsteemiks**.

Definitsioon 4. Süsteemi $P = (X,\pi_X)$ nimetatakse **monotoonseks**, kui suvalise $a \in X\backslash\{c\}$, $c \in X$ korral $\pi_{X'\backslash\{c\}}(a) \le \pi_{X'}(a)$, kus X' on suvaline hulga X alamhulk.

Definitsioon 5. Funktsiooni Q, mis seab igale monotoonse süsteemi P alamhulgale $X' \subseteq X$ vastavusse mittenegatiivse väärtuse $Q(X') = \min_{a \in X'} \pi_{X'}(a)$, nimetatakse **sihifunktsiooniks**.

Definitsioon 6. Monotoonse süsteemi $P = (X,\pi_X)$ allsüsteemi $P^* = (W,\pi_W)$, millel sihifunktsioon Q saavutab oma maksimaalse väärtuse $Q(W) = \max_{X \subseteq X'} Q(X') = \max_{X' \subseteq X} \min_{a \in X'} \pi_{X'}(a)$, nimetatakse süsteemi P **tuumaks**; väärtust Q(W) nimetatakse **tuuma kvaliteedi näitajaks**.

Monotoonse süsteemi ehitamine andmetabelile

Järgnevalt kirjeldame, kuidas andmetabelile ehitada monotoonset süsteemi. Siin eeldame, et meil on lähtesüsteemina vaatluse all andmetabel X(N,M), kus iga element X_{ij} võib omada diskreetset väärtust vahemikus h_j=0,1,...,K_j-1.

Monotoonse süsteemi ehitamiseks:

1. Määratakse sobiv kaalufunktsioon $\pi(X_{ij})$.

2. Määratakse elementidele rakendatavad tegevused (+ või -) ning kaalu ümberarvutamise eeskiri. Miinustegevuse (-) korral toimub elemendi elimineerimisel andmetabelist temaga seotud elementide kaalude vähendamine, plusstegevuse (+) korral aga suurendamine. Plusstegevuse korral aset leidvat kaalude suurenemist saame põhjendada järgmiselt: toimuks nagu uuritava elemendi lisamine (dubleerimine) andmetabelisse (fiktiivselt), mis põhjustabki temaga seotud elementide kaalu suurenemist. Sisuliselt tähendab see seda, et me võimendame teatud omadusega elemente.

3. Rakendatav tegevus (+ või -) ja kaalu ümberarvutamise eeskiri on seotud selliselt, et nad tagaksid süsteemi monotoonsuse. St et suvalisele elemendile $a \in X$ rakendatud tegevus (+ või -) põhjustab kõigil sellistel elementidel $b \in X$, mis on seotud elemendiga a, kaalu $\pi_X(b)$ muutust samas suunas (+ kasvatab, - kahandab). Kui a ja b pole seotud, siis kaalu muutus = 0.

Eelpool kirjeldatud kolm eeldust võimaldavad defineerida praktiliselt lõpmatu hulga erinevaid monotoonseid kaalufunktsioone. Milliste omadustega need olema peavad, see sõltub juba konkreetsest ülesandest ja püstitatud eesmärkidest.

Võimalused monotoonse süsteemi ehitamisel

Monotoonse süsteemi ehitamisel on enamlevinud järgmised moodused. Elementidena, millele rakendatakse teatud (+ või -) tegevusi, vaadeldakse:

1) tabeli elemente X_{ij}: $X=\{X_{ij}\}$, i=1,...,N; j=1,...,M, $W=\{X_{ij}\}$, $W \subseteq X$. Sel juhul tuuma elementide arv $|W| \le N*M$;

2) tabeli ridu: $X=\{X_i\}$, $W=\{X_i\}$, $W \subseteq X$, $|W| \le N$;

3) tabeli veerge: $X=\{X_j\}$, $W=\{X_j\}$, $W \subseteq X$, $|W| \le M$;

4) tabeli ridu ja veerge: $X=\{X_i \cup X_j\}$, $W=\{X_i \cap X_j\}$, $W \subseteq X$, $|W| \le N*M$.

Olenevalt valitud moodusest omab tuum W kuju:

1) „laik" või ristkülik,
2) horisontaalne riba,
3) vertikaalne riba,
4) „rist" või ristkülik.

Loetletud kujudest vajab selgitusi ilmselt „laik". Tema iseärasuseks on elementide erinev arv tuuma eri ridades ja eri veergudes. Seetõttu ta kuju on ebamäärane ja sellest ka nimetus. Mullati baasalgoritmist täiendavate hinnangukriteeriumite rakendamise poolest. Lähemalt käsitleme seda Mullati algoritmi (1971) juures (4.2 Täiendavad võimalused tuuma mõiste määratlemisel).

Algtabel

```
3 3 3 3 3 3 3 1 1 1 1 1 3 2
3 3 3 3 3 3 3 1 1 1 1 1 1 2
3 3 3 3 3 3 3 3 1 1 1 1 1 1
3 3 3 3 3 3 1 3 1 1 1 1 1 1
3 3 3 3 3 3 1 3 1 1 1 1 1 3
3 3 3 3 3 3 2 1 1 1 1 1 1 1
3 2 3 3 3 3 2 1 1 1 1 2 1 1
1 2 3 3 3 5 1 3 1 1 3 2 1 3
1 1 3 3 3 2 2 2 1 1 3 2 1 2
1 1 3 3 3 2 1 1 1 5 3 3 3 3
1 1 3 3 3 2 1 1 1 2 3 3 3 3
1 1 3 3 3 1 1 1 1 2 3 2 3 3
2 1 1 1 1 1 2 1 1 3 2 3 3 3
2 1 1 1 1 1 1 1 1 3 2 3 3 3
1 1 2 2 2 1 1 1 1 2 3 3 3 3
1 1 2 2 2 2 1 1 1 5 3 3 3 3
1 1 2 2 2 3 3 3 1 1 2 1 1 1
1 2 2 2 2 3 3 3 1 1 1 1 1 1
3 1 2 2 2 3 3 3 1 1 1 1 1 1
```

Laik

```
1 * * * 1 * 1 * * 1 1 * * 1
1 * * * 1 * * * * 1 1 * 1 1
* * * * * * * * * * * * * *
1 * * * 1 * 1 * * 1 * 1 * *
* * * * * * * * * * * * * *
1 * * * * 1 1 * * * * 1 1 *
1 * * * 1 * * * * 1 1 1 * 1
1 * * * 1 * 1 * * 1 1 * * 1
1 * * * * 1 * * * 1 1 * * 1
1 * * * 1 * * * * 1 * 1 1 *
1 * * * * 1 * * * 1 1 * * 1
1 * * * * 1 1 * * * * 1 1 *
1 * * * * 1 1 * * * * 1 1 *
1 * * * * 1 1 * * * * 1 1 *
1 * * * 1 1 * * * 1 1 * * 1
1 * * * 1 * * * * 1 1 * * 1
* * * * * * * * * * * * * *
1 * * * 1 * * * * 1 * 1 1 1
1 * * * * 1 1 * * * * 1 1 *
```

Ristkülik

```
* 3 3 3 * * * * * * * * * *
* * * * * * * * * * * * * *
* * * * * * * * * * * * * *
* 3 3 3 * * * * * * * * * *
* 3 3 3 * * * * * * * * * *
* 3 3 3 * * * * * * * * * *
* * * * * * * * * * * * * *
* 3 3 3 * * * * * * * * * *
* 3 3 3 * * * * * * * * * *
* 3 3 3 * * * * * * * * * *
* 3 3 3 * * * * * * * * * *
* * * * * * * * * * * * * *
* * * * * * * * * * * * * *
* 3 3 3 * * * * * * * * * *
* 3 3 3 * * * * * * * * * *
* 3 3 3 * * * * * * * * * *
* * * * * * * * * * * * * *
* * * * * * * * * * * * * *
* 3 3 3 * * * * * * * * * *
```

Vertikaalne riba

```
* 3 3 3 * * * * * * * * * *
* 2 2 2 * * * * * * * * * *
* 2 2 2 * * * * * * * * * *
* 3 3 3 * * * * * * * * * *
* 3 3 3 * * * * * * * * * *
* 3 3 3 * * * * * * * * * *
* 2 2 2 * * * * * * * * * *
* 3 3 3 * * * * * * * * * *
* 3 3 3 * * * * * * * * * *
* 3 3 3 * * * * * * * * * *
* 3 3 3 * * * * * * * * * *
* 1 1 1 * * * * * * * * * *
* 1 1 1 * * * * * * * * * *
* 3 3 3 * * * * * * * * * *
* 3 3 3 * * * * * * * * * *
* 3 3 3 * * * * * * * * * *
* 2 2 2 * * * * * * * * * *
* 2 2 2 * * * * * * * * * *
* 3 3 3 * * * * * * * * * *
```

Horisontaalne riba

```
* * * * * * * * * * * * * *
* * * * * * * * * * * * * *
* * * * * * * * * * * * * *
* * * * * * * * * * * * * *
* * * * * * * * * * * * * *
* * * * * * * * * * * * * *
* * * * * * * * * * * * * *
* * * * * * * * * * * * * *
* * * * * * * * * * * * * *
* * * * * * * * * * * * * *
* * * * * * * * * * * * * *
2 1 1 1 1 1 2 1 1 3 2 3 3 3
2 1 1 1 1 1 1 1 1 3 2 3 3 3
* * * * * * * * * * * * * *
* * * * * * * * * * * * * *
* * * * * * * * * * * * * *
* * * * * * * * * * * * * *
* * * * * * * * * * * * * *
* * * * * * * * * * * * * *
```

II MONOTOONSETE SÜSTEEMIDE RAKENDUSED

Järgnevalt kirjeldame terve rea monotoonsetel süsteemidel baseeruvaid algoritme, mis on leidnud kasutamist praktikas. Nende põhiliseks kasutusvaldkondadeks on andmeanalüüs, tehisintellekt, ekspertsüsteemid, graafiteooria, Boole'i algebra jne.

Andmetabelite korrastamine

Andmeanalüüsis on sagedaseks ülesandeks osata hinnata kogutud andmestu käitumist, ilma et me oleksime eelnevalt kasutanud mingeid tõsisemaid andmeanalüüsi meetodeid (korrelatsioon-, klaster-, faktoranalüüs jt). See on vajalik selleks, et püstitada tööhüpoteese eesmärgiga selgitada, millised objektid milliste tunnuste korral käituvad ühtemoodi.

Klassikaline lähenemine tööhüpoteeside püstitamisel oleks järgmine: kõigepealt tehakse kirjeldavat analüüsi, mille käigus väljastatakse tunnuste sagedusribad ja elementaarstatistikud. Seejärel tellitakse olulisemate (uurija seisukohalt) tunnuste vahelisi risttabeleid. Sellise lähenemise suureks puuduseks on, et me saame niiviisi jälgida tunnuste kooslusi ainult kahe, parimal juhul ka kolme kaupa, suuremaarvulised tunnuste kooslused jäävad vaatluse alt välja.

Käesolevas peatükis kirjeldame terve rea meetodeid, mis võimaldavad lähteandmetabeleid korrastada nii, et selles sisalduvad kõige tüüpilisemad ja kõige omanäolisemad andmekooslused muutuvad nähtavaks. Seejuures ei pea me omama mittemingisugust eelnevat informatsiooni andmete käitumise kohta.

Eeldame, et meil on antud lähteandmetabel X(N,M), kus N on tabeli ridade (objektide) arv, M on tabeli veergude (tunnuste) arv. Omagu iga tunnus j, j = 1,2,...,M, diskreetseid väärtusi h_j = 0,1,...,K_j-1.

Oletame, et meil on vaatluse all andmetabel X(6,5):

i \ j	*1*	*2*	*3*	*4*	*5*
1.	1	0	0	0	0
2.	0	1	0	1	1
3.	0	1	0	1	1
4.	1	1	0	1	0
5.	0	0	1	0	1
6.	0	1	1	1	1

Andmetabeli esimene tunnus tähistab sugu (0 – mees, 1 – naine), tunnus 2 korteri omamist (0 – ei oma, 1 – omab), tunnus 3 haridust (0 – keskharidus, 1 – kõrgem haridus), tunnus 4 aktiivsust (0 – pole aktiivne, 1 – on aktiivne), tunnus 5 auto omamist (0 – ei oma, 1 – omab).

Seda andmetabelit kasutame käesoleva peatükis kõikide käsitletavate korrastusmeetodite selgitamisel.

Mida tähendab andmetabeli korrastamine? Eelpool kirjeldatud ülesande kontekstis tähendab see seda, et me peaksime leidma mõõdu, mille alusel saaksime hinnata objektide/tunnuste lähedust. Korrastuse tulemusena peaksid selle mõõdu järgi lähedased objektid paiknema lähestikku. Nagu edaspidi näeme, on selliseid mõõte võimalik defineerida mitmeid, samamoodi esineb ka mitmeid erinevaid korrastustehnikaid.

Järgnevalt kirjeldame professor Leo Võhandu poolt välja töötatud erinevaid mõõtusid ja tehnikaid andmetabelite korrastamiseks (Võhandu et al. 2006).

1 Konformismiskaala

Üheks võimaluseks objektide läheduse hindamisel on lähtuda teda kirjeldavate tunnuste väärtuste tüüpilisusest. Sellel baseerubki järgnevalt kirjeldatav skaala nimega „konformismiskaala".

Algoritm

Samm 1. Leiame igale tunnusele j tema väärtuste hj esinemissagedused Z_{hj} andmetabelis X(N,M).

Samm 2. Asendame andmetabeli iga elemendi X_{ij} (i=1,...,N, j=1,...,M) tema esinemissagedusega Z_{ij}. Saame andmetabeli Z(N,M).

Samm 3. Leiame andmetabeli Z igale reale i tema liikmete summa S_i.

Samm 4. Teostame tabeli ridade ümberjärjestamise (kahanevalt) vastavalt summadele S_i.

Summa S_i kujutabki endast objekti väärtust konformismiskaalal. Võib tekkida küsimus, et mida see suurus S_i sisuliselt näitab? Vastuseks on: mida suurem see on, seda konformsem (tüüpilisem) on antud objekt, mida väiksem on see summa, seda omanäolisem on see objekt. Kui objekt omab suuremaid sagedusi, siis omab ta tüüpilisemaid (enimlevinumaid) tunnuste väärtusi antud andmetabelis. Kui ta omab väiksemaid sagedusi, siis omab ta vähem esinevaid tunnuste väärtusi antud andmetabelis.

Konformismiskaala korral saame leida ka antud andmetabeli suhtes minimaalse või maksimaalse konformismi, st selle skaala algus ja lõpp-punkti antud andmetabeli X jaoks. Selleks on vaja summeerida kõikide tunnuste korral vastavalt minimaalsed või maksimaalsed sagedused.

Konformismiskaala arvutuslik keerukus on O(MN) operatsiooni.

Näide

Oletame, et meil on andmetabel X(6,5):

i \ j	*1*	*2*	*3*	*4*	*5*
1.	1	0	0	0	0
2.	0	1	0	1	1
3.	0	1	0	1	1
4.	1	1	0	1	0
5.	0	0	1	0	1
6.	0	1	1	1	1

Rakendame eelpool kirjeldatud algoritmi.

Samm1. Leiame väärtuste 0 ja 1 esinemissagedused.

0	4	2	4	2	2
1	2	4	2	4	4

Sammud 2 ja **3**. Teeme sagedusteisenduse ja leiame reasummad.
Oletame, et meil on vaatluse all andmetabel X(6,5):

i \ j	*1*	*2*	*3*	*4*	*5*	S_i
1.	2	2	4	2	2	12
2.	4	4	4	4	4	20
3.	4	4	4	4	4	20
4.	2	4	4	4	2	16
5.	4	2	2	2	4	14
6.	4	4	2	4	4	18

Samm 4. Järjestame lähtetabeli read kahanevalt vastavalt väärtusele S_i.

i \ j	*1*	*2*	*3*	*4*	*5*	S_i
2.	4	4	4	4	4	20
3.	4	4	4	4	4	20
6.	4	4	2	4	4	18
4.	2	4	4	4	2	16
5.	4	2	2	2	4	14
1.	2	2	4	2	2	12

Nagu näeme, on andmetabel muutunud paremini loetavaks kui enne. Ei maksa unustada, et me oleme andmetabeli korrastanud vaid ridu pidi. Korrastades ta veerge pidi, muutub ta veelgi informatiivsemaks.

Veergude korrastamiseks on mitu võimalust. Kirjeldame siin ühte võimalikku varianti.
Võtame aluseks sagedusteisenduse ja summeerime veerule j vastavad sageduste ruudud:

$S_j = \sum Z_{ij}^2$, i=1,...,N.

Mida suurem on summa S_j, seda konformsem (homogeensem, väiksema varieeruvusega) on antud tunnus.
Meie näite korral saame tulemuseks
$S_1=72$, $S_2=72$, $S_3=72$, $S_4=72$, $S_5=72$.

Nagu näeme, on kõik suurused Sj võrdsed ja selle mõõdu kasutamine tabeli veergude korrastamiseks ei garanteeri soovitud tulemust. Antud näide on väga õpetlik just selle poolest, et valitud mõõt võib osutuda liiga „nõrgaks" analüüsitava andmetabeli sisemise struktuuri seisukohalt, ta ei suuda tabelit lahutada homogeenseteks allosadeks.

Konformismiskaala ei välista võimalust, et erineva struktuuriga objektid andmetabelis võivad omada ühesugust või lähedast väärtust – skaala on liiga „nõrk" nende objektide eristamiseks. Järgnevalt kirjeldame uut mõõtskaalat nimega mõjuskaala, mille lahutusvõime on oluliselt parem konformismiskaalast. Algoritmi samm-sammulist tööd demonstreerib ka video: konformismiskaala.

2 Mõjuskaala

Oletame, et oleme teinud sagedusteisenduse $X_{ij} \rightarrow Z_{ij}$ ning et andmetabeli iga objekti i varieeruvus on kirjeldatud talle vastavate tunnuste väärtuste ruutude summaga

$$S_i = \sum Z_{ij}^2,\ j=1,...,M,$$

siis on kogu süsteemi varieeruvus

$$S=\sum S_i = \sum\sum Z_{ij}^2,\ i=1,...,N,\ j=1,...,M.$$

Nüüd saame määratleda **i-nda objekti mõju** kui ruutude summa, mille võrra väheneb süsteemi koguvarieeruvus S, kui objekt i lülitada analüüsist välja.

Objekti elimineerimine tähendab talle vastavate elementide X_{ij} väljalülitamist. Sellega kaasneb aga elemendile X_{ij} vastava tunnuse väärtuse h_j esinemissageduse vähenemine ühe võrra. Nüüd saame mõõta elemendi X_{ij} elimineerimisest tingitud süsteemi koguvarieeruvuse muutust G_{ij}. Kuna tunnuse j väärtus h_j pärast objekti i elimineerimist omab veel Z_{hj-1} samasugust väärtust (h_j), siis süsteemi uus varieeruvus ühe elemendi korral võrdub

$$G_{ij} = (Z_ih_j-1)(Z_ih_{ij}^2 - (Z_ih_{j-1})^2) = 2Z_ih_j^2 - 3Z_ih_j + 1.$$

Teades elementide mõjusid, saame arvutada objekti mõju

$$G_i = \sum G_{ij},\ j=1,...,M.$$

Saadud väärtuste alusel järjestame andmetabeli read.
Samamoodi võime käituda ka tabeli veergude suhtes. Arvutame igale veerule j suuruse

$$G_j = \sum G_{ij},\ i=1,...,N.$$

Saadud summade G_j alusel järjestame tabeli veerud.
Mõjuskaala rakendamise algoritm on järgmine.

Algoritm

Samm 1. Leiame igale tunnusele j tema väärtuste hj esinemissagedused Z_jh_j andmetabelis X(N,M).
Samm 2. Arvutame andmetabeli igale elemendile X_{ij} tema mõju G_{ij}. Leiame nende alusel kõik reasummad G_i.
Samm 3. Leiame iga rea igale elemendile X_{ij} tema väärtuste h_j esinemissagedused Z_ih_j andmetabelis X(N,M).
Samm 4. Arvutame andmetabeli igale elemendile X_{ij} reasageduste Z_ih_j alusel tema mõju G_{ij}. Leiame nende alusel kõik veerusummad G_j.
Samm 5. Teostame tabeli ridade ja veergude ümberjärjestamise (kahanevalt) vastavalt summadele G_i ja G_j.

Näide

Kasutame eelmise näite andmetabelit

i \ j	*1*	*2*	*3*	*4*	*5*
1.	1	0	0	0	0
2.	0	1	0	1	1
3.	0	1	0	1	1
4.	1	1	0	1	0
5.	0	0	1	0	1
6.	0	1	1	1	1

Rakendame eelpool kirjeldatud algoritmi.

Samm1. Leiame väärtuste 0 ja 1 esinemissagedused $Z_j h_j$.

0	4	2	4	2	2
1	2	4	2	4	4

Samm 2. Arvutame elementidele X_{ij} mõjud G_{ij}.

$X_{11}=1$, $Z_{11}=2$, $G_{11}=(2-1)(2^2 - (2-1)^2)=1(4-1)=3$
$X_{12}=0$, $Z_{12}=2$, $G_{12}=(2-1)(2^2 - (2-1)^2)=1(4-1)=3$
$X_{13}=0$, $Z_{13}=4$, $G_{13}=(4-1)(4^2 - (4-1)^2)=3(16-9)=21$
...

Esitame elementidele vastavad mõjud tabelina. Selle ääresummadeks on tabeli ridadele ja veergudele vastavad mõjud.

i \ j	*1*	*2*	*3*	*4*	*5*	G_i	Järjestus
1.	3	3	21	3	3	33	6.
2.	21	21	21	21	21	105	1.
3.	21	21	21	21	21	105	2.
4.	3	21	21	21	3	69	4.
5.	21	3	3	3	21	51	5.
6.	21	21	3	21	21	87	3.

Pärast korrastamist saame tulemuseks

i \ j	*1*	*2*	*3*	*4*	*5*	G_i	Järjestus
2.	21	21	21	21	21	105	1.
3.	21	21	21	21	21	105	2.
6.	21	21	3	21	21	87	3.
4.	3	21	21	21	3	69	4.
5.	21	3	3	3	21	51	5.
1.	3	3	21	3	3	33	6.

Samm 3. Leia igale väärtusele X_{ij} tema esinemissagedus $Z_i h_j$ reas i.

i \ j	*1*	*2*	*3*	*4*	*5*
1.	1	0	0	0	0
2.	0	1	0	1	1
3.	0	1	0	1	1
4.	1	1	0	1	0
5.	0	0	1	0	1
6.	0	1	1	1	1

0	1
4	1
2	3
2	3
2	3
3	2
1	4

Tehes sagedusteisenduse $X_{ij} \rightarrow Z_i h_j$ (mõttes, sest elemendi mõju saab arvutada lähtudes elemendi sagedusest, seega vahepealse tabeli tekitamine on üleliigne töö!), saaksime tabeli

i \ j	*1*	*2*	*3*	*4*	*5*
1.	1	4	4	4	4
2.	2	3	2	3	3
3.	2	3	2	3	3
4.	3	3	2	3	2
5.	3	3	2	3	2
6.	1	4	4	4	4

Samm 4. Arvutame elementide X_{ij} mõjud G_{ij} lähtudes reasagedustest Z_ih_j.

$X_{11}=1$, $Z_{11}=1$, $G_{11}=(1-1)(1^2 - (1-1)^2)=0$
$X_{12}=0$, $Z_{12}=4$, $G_{12}=(4-1)(4^2 - (4-1)^2)=3(16-9)=21$
...
$X_{21}=0$, $Z_{21}=2$, $G_{21}=(2-1)(2^2 - (2-1)^2)=1(4-1)=3$
$X_{22}=1$, $Z_{22}=3$, $G_{22}=(3-1)(3^2 - (3-1)^2)=2(9-4)=10$
...

Esitame elementidele vastavad mõjud tabelina. Selle tabeli veerusummadeks on tabeli veergudele (tunnustele) vastavad mõjud. Saame tabeli

i \ j	*1*	*2*	*3*	*4*	*5*
1.	0	21	21	21	21
2.	3	10	3	10	10
3.	3	10	3	10	10
4.	10	10	3	10	3
5.	10	10	3	10	3
6.	0	21	21	21	21
Gj	26	82	54	82	68
Järjestus	5.	1.	4.	2.	3.

Samm 5. Korrastame andmetabeli saadud rea- ja veerumõjude alusel. Saame tabeli:

i \ j	*2*	*4*	*5*	*3*	*1*	G_i	Järjestus
2.	1	1	1	0	0	105	1.
3.	1	1	1	0	0	105	2.
6.	1	1	1	1	0	87	3.
4.	1	1	0	0	1	69	4.
5.	0	0	1	1	0	51	5.
1.	0	0	0	0	1	33	6.
Gj	82	82	68	54	26		
Järjestus	1.	2.	3.	4.	5.		

Nagu näeme, on tabel muutunud oluliselt informatiivsemaks. Tabeli ülemises vasakus nurgas paiknevad süsteemi kui terviku seisukohalt kõige tüüpilisemad elemendid. Tabeli allosas paiknevad seevastu kõige omanäolisemad elemendid (vaadake tunnuste ja väärtuste sisu ning püüdke ise interpreteerida ja järeldada!). Algoritmi samm-sammulist tööd demonstreerib ka video: mõjuskaala.

Kuidas on seotud konformismi- ja mõjuskaala?

Mõju- ja konformismiskaala on vahetult seotud, ühe väärtused on teisest tuletatavad.

$G_i = \sum G_i = \sum(2Z_ih_j^2 - 3Z_ih_j + 1)=2\sum Z_ih_j^2 - 3\sum Z_ih_j + M = 2S_i - 3K_i - M.$

Saadud avaldises tähistab S_i i-nda objekti varieeruvust, K_i i-nda objekti konformsust ja M tunnuste arvu (veerge andmetabelis).
Arvestades, et mõjufunktsioon G_{ij} on ruutfunktsioon, siis ta võimendab neid objekte, mis on konformsemad.
Mõjuskaala arvutuslik keerukus on O(MN) operatsiooni.

Monotoonsete süsteemide elementaartehnikad

3 Andmetabeli korrastamine: miinustehnika, plusstehnika ja segatehnika

Eelmises alajaotuses kirjeldasime ühe klassi keerukamat kaalufunktsiooni – nn mõjuskaalat. Selle kasutamine andis võrreldes konformismiskaalaga paremaid tulemusi, sest tegemist on ruut-funktsiooniga ja seetõttu on tema võime objekte eristada suurem.

Käesolevas peatükis kirjeldame kahte monotoonsete tegevuste gruppi: miinus- ja plusstehnikat. Need demonstreerivad eriti ilmekalt, kuidas on omavahel seotud kaalufunktsiooni väärtus ja andmetabeli elementidele rakendatavad tegevused (elementide eemaldamine, elementide lisamine).

3.1 Miinustehnika

Kirjeldatav tehnika baseerub jällegi sagedusteisendusel. Idee seisneb objektide järjestikusel eemaldamisel andmetabelist ja analüüsi jäänud objektidele uute kaalude arvutamises. Erinevalt mõjuskaalast, kus i-nda objekti eemaldamisel jäi analüüsi N-1 objekti, jääb siin analüüsi N-i objekti.

Algoritm

Samm 1. Leiame igale tunnusele j (objektile i) tema väärtuste h_j (elementide X_{ij}) esinemissagedused Z_jh_j (Z_ih_j) andmetabelis X(N,M) ja arvutame igale objektile (tunnusele) tema konformsuse (kaalu).

Samm 2. Elimineerime objekti (tunnuse), mis omab **väikseimat** kaalu. Objekti (tunnuse) elimineerimine põhjustab temaga seotud objektide (tunnuste) kaalude **vähenemist**.

Samm 3. Analüüsi jäänud objektidele (tunnustele) arvutatakse uued kaalud järgmise eeskirja alusel: leitakse analüüsis oleva objekti i (tunnuse j) ja elimineeritava objekti (tunnuse) positsiooniliselt ühesugust väärtust omavate elementide arv A.
i-nda objekti (j-nda tunnuse) uus kaal $S_{uus}=S_{vana}$ **-** A.

Samm 4. Kui analüüsis on objekte (tunnuseid), mine Samm 2.

Samm 5. Rakendame Samme 1 kuni 5 andmetabeli veergudele (tunnustele).

Samm 6. Võttes aluseks objektide ja tunnuste elimineerimise järjekorra, korrastame andmetabeli read ja veerud.

Näide

Kasutame eelmise näite andmetabelit

i \ j	*1*	*2*	*3*	*4*	*5*
1.	1	0	0	0	0
2.	0	1	0	1	1
3.	0	1	0	1	1
4.	1	1	0	1	0
5.	0	0	1	0	1
6.	0	1	1	1	1

Rakendame eelpool kirjeldatud algoritmi kõigepealt objektidele.

Samm1. Leiame väärtuste 0 ja 1 esinemissagedused Z_jh_j.

0	4	2	4	2	2
1	2	4	2	4	4

Arvutame objektidele kaalud.

i \ *j*	*1*	*2*	*3*	*4*	*5*	S_i
1.	2	2	4	2	2	12
2.	4	4	4	4	4	20
3.	4	4	4	4	4	20
4.	2	4	4	4	2	16
5.	4	2	2	2	4	14
6.	4	4	2	4	4	18

Samm 2. Vähimat kaalu omab esimene objekt: kaal=12. Elimineerime selle.

Samm 3. Arvutame allesjäänud objektidele uued kaalud.

1. 1 0 0 0 0	*1.* 1 0 0 0 0	*1.* 1 0 0 0 0	*1.* 1 0 0 0 0	*1.* 1 0 0 0 0
2. 0 1 0 1 1	*3.* 0 1 0 1 1	*4.* 1 1 0 1 0	*5.* 0 0 1 0 1	*6.* 0 1 1 1 1
* * 0 * *	* * 0 * *	1 * 0 * 0	* 0 * 0 *	* * * * *

Kokkulangevaid elemente (A) on järgnevalt:

objekt i:	2	3	4	5	6
A:	1	1	3	2	0

Arvutame uued kaalud:

objekt i:	2	3	4	5	6
S_{vana}:	20	20	16	14	18
- A:	1	1	3	2	0
S_{uus}:	19	19	13	12	18

Samm 4. Analüüsi on jäänud 5 objekti. Mine Samm 2.

Samm 2. Jne.

Järgnevalt esitame algoritmi kogu töö tabeli kujul.

i \ *j*	*1*	*2*	*3*	*4*	*5*	S_i	Kaal						Järjestus
1.	2	2	4	2	2	12	–						1.
2.	4	4	4	4	4	20	19	17	14	10	–		5.
3.	4	4	4	4	4	20	19	17	14	10	5	–	6.
4.	2	4	4	4	2	16	13	13	–				3.
5.	4	2	2	2	4	14	12	–					2.
6.	4	4	2	4	4	18	18	15	13	–			4.

Samm 5. Rakendame nüüd algoritmi samme Samm 1 kuni Samm 5 tunnustele.

Samm 1. Leia igale väärtusele X_{ij} tema esinemissageduse Z_ih_j reas i.

i \ *j*	*1*	*2*	*3*	*4*	*5*
1.	1	0	0	0	0
2.	0	1	0	1	1
3.	0	1	0	1	1
4.	1	1	0	1	0
5.	0	0	1	0	1
6.	0	1	1	1	1

0	1
4	1
2	3
2	3
2	3
3	2
1	4

Arvutame tunnustele nende kaalud.

i \ j	*1*	*2*	*3*	*4*	*5*
1.	1	4	4	4	4
2.	2	3	2	3	3
3.	2	3	2	3	3
4.	3	3	2	3	2
5.	3	3	2	3	2
6.	1	4	4	4	4
Sj	12	20	16	20	18

Samm 2. Elimineerime tunnuse 1 kui nõrgima: kaal=12.
Samm 3. Leiame analüüsi jäänud tunnustele kokkulangevate elementide arvu elimineeritava tunnuse suhtes.

1	*2*		*1*	*3*		*1*	*4*		*1*	*5*	
1	0	*	1	0	*	1	0	*	1	0	*
0	1	*	0	0	0	0	1	*	0	1	*
0	1	*	0	0	0	0	1	*	0	1	*
1	1	1	1	0	*	1	1	1	1	0	*
0	0	0	0	1	*	0	0	0	0	1	*
0	1	*	0	1	*	0	1	*	0	1	*
A		2			2			2			0

Arvutame tunnustele uued kaalud:

Tunnus j:	2	3	4	5
S_{vana}:	20	16	20	18
- A:	2	2	2	0
S_{uus}:	18	14	18	18

Samm 4. Analüüsi on jäänud järele 4 tunnust. Mine Samm 2.

Samm 2. Jne.

Järgnevalt esitame algoritmi kogu töö tabeli kujul:

i \ j	*1*	*2*	*3*	*4*	*5*
1.	1	0	0	0	0
2.	0	1	0	1	1
3.	0	1	0	1	1
4.	1	1	0	1	0
5.	0	0	1	0	1
6.	0	1	1	1	1
Sj	12	20	16	20	18
	–	18	14	18	18
		16	–	16	14
		12		12	–
		–		6	
				–	
Järjestus	1.	4.	2.	5.	3.

Samm 6. Korrastame andmetabeli read ja veerud.

i \ j	*1*	*3*	*5*	*2*	*4*	S_i
1.	1	0	0	0	0	12
5.	0	1	1	0	0	12
4.	1	0	0	1	1	13
6.	0	1	1	1	1	13
2.	0	0	1	1	1	10
3.	0	0	1	1	1	5
Sj	12	14	14	12	6	

Võrreldes lähtetabeliga on korrastatud tabel hulga informatiivsem, sest nähtavaks saavad andmete tüüpilisus ja iseärasused. Kõige homogeensem elementide grupp tekib korrastatud tabeli alumisse paremasse nurka, kõige isepäisem ülesse vasakusse nurka. Lisaks võimaldab miinustehnika ka objekte ja tunnuseid grupeerida. Selleks peame objektide korral liikuma piki kaalusid alt üles, tunnuste korral paremalt vasakule. Kui kaal enam ei kasva, siis algab uute omadustega grupp, st et see objekt või tunnus erineb eelmistest rohkem kui temale järgnevates, seega on tekkinud uus kvaliteet:

i \ j	*1*	*3*	*5*	*2*	*4*	S_i
1.	1	0	0	0	0	12
5.	0	1	1	0	0	12
4.	1	0	0	1	1	13
6.	0	1	1	1	1	13
2.	0	0	1	1	1	10
3.	0	0	1	1	1	5
Sj	12	14	14	12	6	

Selle eeskirja järgi töödeldud tabelist näeme, et objektide osas on leitud 4 gruppi: Go1 (objektid 3,2 ja 6), Go2 (4), Go3 (5), Go4 (1). Tunnuste osas 3 gruppi: Gt1 (tunnused 4, 2 ja 5), Gt2 (3), Gt3 (1).

Kirjeldatud tehnika on tegelikult ammusest ajast olnud kasutusel ja põhjendatud psühholoogide poolt. Nimelt, kui kedagi tuleks vallandada, siis on selleks kollektiiviga kõige vähem seotud või sinna mitte sobiv isik.

Algoritmi samm-sammulist tööd demonstreerib ka video: miinustehnika

3.2 Plusstehnika

Plusstehnika on algoritmiliste tegevuste seisukohalt sarnane miinustehnikale. Põhiline erinevus seisneb uute kaalude arvutamises pärast mingi objekti/tunnuse väljalülitamist:

uus kaal $S_{uus} = S_{vana} + A$.

Teiseks oluliseks erinevuseks oleks see, et alati elimineeritakse kõige suuremat kaalu omav objekt/tunnus.

Muus osas jääb algoritm samaks.

Algoritm

Samm 1. Leiame igale tunnusele j (objektile i) tema väärtuste h_j (elementide X_{ij}) esinemissagedused Z_jh_j (Z_jh_j) andmetabelis X(N,M) ja arvutame igale objektile (tunnusele) tema konformsuse (kaalu).

Samm 2. Elimineerime objekti (tunnuse), mis omab **suurimat** kaalu. Objekti (tunnuse) elimineerimine põhjustab temaga seotud objektide (tunnuste) kaalude **suurenemist**.

Samm 3. Analüüsi jäänud objektidele (tunnustele) arvutatakse uued kaalud järgmise eeskirja alusel: leitakse analüüsis oleva objekti i (tunnuse j) ja elimineeritava objekti (tunnuse) positsiooniliselt ühesugust väärtust omavate elementide arv A.
i-nda objekti (j-nda tunnuse) uus kaal $S_{uus}=S_{vana}+A$.

Samm 4. Kui analüüsis on objekte (tunnuseid), mine Samm 2.

Samm 5. Rakendame Samme 1 kuni 5 andmetabeli veergudele (tunnustele).

Samm 6. Võttes aluseks objektide ja tunnuste elimineerimise järjekorra, korrastame andmetabeli read ja veerud.

Näide

Kasutame eelmise näite andmetabelit

i \ j	*1*	*2*	*3*	*4*	*5*
1.	1	0	0	0	0
2.	0	1	0	1	1
3.	0	1	0	1	1
4.	1	1	0	1	0
5.	0	0	1	0	1
6.	0	1	1	1	1

Rakendame eelpool kirjeldatud algoritmi kõigepealt objektidele.

Samm1. Leiame väärtuste 0 ja 1 esinemissagedused $Z_j h_j$.

0	4	2	4	2	2
1	2	4	2	4	4

Arvutame objektidele kaalud.

i \ j	*1*	*2*	*3*	*4*	*5*	S_i
1.	2	2	4	2	2	12
2.	4	4	4	4	4	20
3.	4	4	4	4	4	20
4.	2	4	4	4	2	16
5.	4	2	2	2	4	14
6.	4	4	2	4	4	18

Samm 2. Suurimat kaalu omab teine objekt: kaal=20. Elimineerime selle.
Samm 3. Arvutame allesjäänud objektidele uued kaalud.

2. 0 1 0 1 1	*2.* 0 1 0 1 1	*2.* 0 1 0 1 1	*2.* 0 1 0 1 1	*2.* 0 1 0 1 1
1. 1 0 0 0 0	*3.* 0 1 0 1 1	*4.* 1 1 0 1 0	*5.* 0 0 1 0 1	*6.* 0 1 1 1 1
* * 0 * *	0 1 0 1 1	* 1 0 1 *	0 * * * 1	0 1 * 1 1

Kokkulangevaid elemente (A) on järgnevalt:

objekt i:	1	3	4	5	6
A:	1	5	3	2	4

Arvutame uued kaalud:

objekt i:	1	3	4	5	6
S_{vana}:	12	20	16	14	18
A:	1	5	3	2	4
S_{uus}:	13	25	19	16	22

Samm 4. Analüüsi on jäänud 5 objekti. Mine Samm 2.

Samm 2. Jne.

Järgnevalt esitame algoritmi kogu töö tabeli kujul.

i / j	*1*	*2*	*3*	*4*	*5*	S_i	Kaal						Järjestus
1.	2	2	4	2	2	12	13	14	14	17	19	–	6.
2.	4	4	4	4	4	20	–						1.
3.	4	4	4	4	4	20	25	–					2.
4.	2	4	4	4	2	16	19	22	24	–			4.
5.	4	2	2	2	4	14	16	18	21	21	–		5.
6.	4	4	2	4	4	18	22	26	–				3.

Samm 5. Rakendame nüüd algoritmi samme Samm 1 kuni Samm 5 tunnustele.

Samm 1. Leia igale väärtusele X_{ij} tema esinemissageduse reas i Z_ih_j.

i \ j	*1*	*2*	*3*	*4*	*5*
1.	1	0	0	0	0
2.	0	1	0	1	1
3.	0	1	0	1	1
4.	1	1	0	1	0
5.	0	0	1	0	1
6.	0	1	1	1	1

0	1
4	1
2	3
2	3
2	3
3	2
1	4

Arvutame tunnustele nende kaalud.

i \ j	*1*	*2*	*3*	*4*	*5*
1.	1	4	4	4	4
2.	2	3	2	3	3
3.	2	3	2	3	3
4.	3	3	2	3	2
5.	3	3	2	3	2
6.	1	4	4	4	4
S_j	12	20	16	20	18

Samm 2. Elimineerime tunnuse 2 kui tugevaima: kaal=20.

Samm 3. Leiame analüüsi jäänud tunnustele kokkulangevate elementide arvu elimineeritava tunnuse suhtes.

2	*1*		*2*	*3*		*2*	*4*		*2*	*5*	
0	1	*	0	0	0	0	0	0	0	0	0
1	0	*	1	0	*	1	1	1	1	1	1
1	0	*	1	0	*	1	1	1	1	1	1
1	1	1	1	0	*	1	1	1	1	0	*
0	0	0	0	1	*	0	0	0	0	1	*
1	0	*	1	1	1	1	1	1	1	1	1
A		2			2			6			4

Arvutame tunnustele uued kaalud:

Tunnus j:	2	3	4	5
S_{vana}:	20	16	20	18
A:	2	2	6	4
S_{uus}:	22	18	26	22

Samm 4. Analüüsi on jäänud 4 tunnust. Mine Samm 2.

Samm 2. Jne.

Järgnevalt esitame algoritmi kogu töö tabeli kujul.

i \ j	*1*	*2*	*3*	*4*	*5*
1.	1	0	0	0	0
2.	0	1	0	1	1
3.	0	1	0	1	1
4.	1	1	0	1	0
5.	0	0	1	0	1
6.	0	1	1	1	1
Sj	12	20	16	20	18
	14	–	18	26	22
	16		20	–	26
	16		24		–
	18		–		
	–				
Järjestus	5.	1.	4.	2.	3.

Samm 6. Korrastame andmetabeli read ja veerud.

i \ j	*2*	*4*	*5*	*3*	*1*	Kaal i
2.	1	1	1	0	0	20
3.	1	1	1	0	0	25
6.	1	1	1	1	0	26
4.	1	1	0	0	1	24
5.	0	0	1	1	0	21
1.	0	0	0	0	1	19
Kaal j	20	26	26	24	18	

Võrreldes lähtetabeliga on korrastatud tabel palju informatiivsem, sest nähtavaks saavad andmete tüüpilisus ja iseärasused. Kõige homogeensem elementide grupp tekib korrastatud tabeli ülemisse vasakusse nurka, kõige isepäisem alla paremasse nurka. Lisaks võimaldab plusstehnika ka objekte ja tunnuseid grupeerida. Selleks peame objektide korral liikuma piki kaalusid ülalt alla, tunnuste korral vasakult paremale. Kui kaal enam ei kasva, siis algab uute omadustega grupp – see objekt või tunnus erineb eelmistest rohkem kui temale järgnevates ning seega on tekkinud uus kvaliteet:

i \ j	*2*	*4*	*5*	*3*	*1*	Kaal i
2.	1	1	1	0	0	20
3.	1	1	1	0	0	25
6.	1	1	1	1	0	26
4.	1	1	0	0	1	24
5.	0	0	1	1	0	21
1.	0	0	0	0	1	19
Kaal j	20	26	26	24	18	

Selle eeskirja järgi töödeldud tabelist näeme, et objektide osas on leitud 4 gruppi: Go1 (objektid 2,3 ja 6), Go2 (4), Go3 (5), Go4 (1). Tunnuste osas samuti 4 gruppi: Gt1 (tunnused 2 ja 4), Gt2 (5), Gt3 (3), Gt4 (1).

Kirjeldatud tehnika on tegelikult ammusest ajast olnud rakendusel ja põhjendatud psühholoogide poolt. Nimelt, kui kollektiivis töö liigub vales suunas, siis tuleks vallandada vastava suuna liider. Ja läbi plusstehnika võimendatakse neid objekte (isikuid), kes on liidriga lähedalt seotud, st on temaga sarnased. See teeb võimalikuks järgmisena elimineerida eelmise liidri võimaliku järglase.

Algoritmi samm-sammulist tööd demonstreerib ka video: plusstehnika.

3.3 Segatehnika

Segatehnika erineb algoritmiliste tegevuste seisukohalt miinus- ja plusstehnikast. Põhiline erinevus seisneb uute kaalude arvutamise mehhanismis ja otsustamises, milline objekti/tunnus järgmisena välja lülitada. Meetod eeldab, et objektidele/tunnustele on leitud algselt mingid kaalud, näiteks konformismiskaalal selleks juhuks, kui ei suudeta valida, milline objekt esimesena välja lülitada. (Tegelikult võib alustada suvalisest objektist. Määrav on, millise objekti suhtes järjestust soovitakse saavutada). Alati eemaldatakse objekt/tunnus, mis on eelmisele eemaldatule kõige sarnasem – väärtuste kokkulangemiste arv on kõige suurem.

Teiseks oluliseks erinevuseks on korrastamisel rakendatav tehnika. Iga järgmisena eemaldatava kandidaadi kaalu ei arvutata lähtudes algtabeli väärtuste esinemissagedustest, vaid korrastamise tulemusena formeeruva objekt/tunnus tabeli väärtuste esinemissagedustest. Kuni pole ühtegi kandidaati eemaldatud, on tulemustabel tühi – kõik sagedused=0. Objektid/tunnused lisanduvad tulemustabelisse ühekaupa, vastavalt hakkavad kasvama (+1) ka lisanduvate väärtuste esinemissagedused. Iga järgmise iteratsiooni objekti/tunnuse kaalude arvutamise aluseks on tulemustabeli hetkeseis. Algoritmi töö lõppemisel on tulemustabeli ja algtabeli väärtuste esinemissagedused võrdsed.

Algoritm

Samm 1. Leiame igale tunnusele j (objektile i) tema väärtuste h_j (elementide X_{ij}) esinemissagedused Z_jh_j (Z_jh_j) andmetabelis X(N,M) ja arvutame igale objektile (tunnusele) tema konformsuse (kaalu).

Samm 2. Elimineerime objekti (tunnuse), mis omab **suurimat** kaalu. Objekti (tunnuse) elimineerimine põhjustab temaga seotud objektide (tunnuste) kaalude **suurenemist**.

Samm 3. Analüüsi jäänud objektidele (tunnustele) arvutatakse uued kaalud järgmise eeskirja alusel: leitakse analüüsis oleva objekti i (tunnuse j) ja elimineeritava objekti (tunnuse) positsiooniliselt ühesugust väärtust omavate elementide arv A.
i-nda objekti (j-nda tunnuse) uus kaal $S_{uus}=S_{vana}$ **+** A.

Samm 4. Kui analüüsis on objekte (tunnuseid), mine Samm 2.

Samm 5. Rakendame Samme 1 kuni 5 andmetabeli veergudele (tunnustele).

Samm 6. Võttes aluseks objektide ja tunnuste elimineerimise järjekorra, korrastame andmetabeli read ja veerud.

Näide

Kasutame eelmise näite andmetabelit

i \ j	*1*	*2*	*3*	*4*	*5*
1.	1	0	0	0	0
2.	0	1	0	1	1
3.	0	1	0	1	1
4.	1	1	0	1	0
5.	0	0	1	0	1
6.	0	1	1	1	1

Rakendame eelpool kirjeldatud algoritmi kõigepealt objektidele.

Samm1. Leiame väärtuste 0 ja 1 esinemissagedused Z_jh_j.

0	4	2	4	2	2
1	2	4	2	4	4

Eeldame näiteks, et esimese välja visatava objekti leiame konformismiskaala alusel (vähima kaaluga objekt). Arvutame objektidele kaalud.

i \ j	*1*	*2*	*3*	*4*	*5*	S_i
1.	2	2	4	2	2	12
2.	4	4	4	4	4	20
3.	4	4	4	4	4	20
4.	2	4	4	4	2	16
5.	4	2	2	2	4	14
6.	4	4	2	4	4	18

Vähimat kaalu omab objekt 1, kaal=12. Moodustame uue sagedustabeli, mille kõik sagedused=0.

Samm 2. Elimineerime objekti. (Kuna uus sagedustabel on tühi, siis esimesena elimineeritava objekti algne kaal S_{vana} uues süsteemis =0, S_{vana}=0+0+0+0+0). Suurendame uues sagedustabelis elimineeritavate väärtuste esinemissagedusi ühe võrra (+1):

0	0	1	1	1	1
1	1	0	0	0	0

Samm 3. Arvutame allesjäänud objektidele väljavisatud objekti suhtes kokkulangevuste arvu ja uued kaalud Suus.

1. 1 0 0 0 0	*1.* 1 0 0 0 0	*1.* 1 0 0 0 0	*1.* 1 0 0 0 0	*1.* 1 0 0 0 0
2. 0 1 0 1 1	*3.* 0 1 0 1 1	*4.* 1 1 0 1 0	*5.* 0 0 1 0 1	*6.* 0 1 1 1 1
* * 0 * *	* * 0 * *	1 * 0 * 0	* 0 * 0 *	* * * * *

Kokkulangevaid elemente (A) on järgnevalt:

objekt i:	2	3	4	5	6
A:	1	1	3	2	0

Arvutame uued kaalud:

objekt i:	2	3	4	5	6
S_{vana}:	0	0	0	0	0
+ A:	1	1	3	2	0
S_{uus}:	1	1	3	2	0

Järgmisena tuleb elimineerida objekt 4, sest tema A väärtus (kokkulangevuste arv) on kõige suurem.

Samm 4. Analüüsi on jäänud 5 objekti. Mine Samm 2.

Samm 2. Elimineerime objekti 4

4.	1	1	0	1	0

Suurendame uues sagedustabelis elimineeritavate väärtuste esinemissagedusi ühe võrra (+1):

0	0	1	1+1=2	1	1+1=2
1	1+1=2	0+1=1	0	0+1=2	0

Samm 3. Arvutame allesjäänud objektidele väljavisatud objekti suhtes kokkulangevuste arvu ja uued kaalud S_{uus}.

4. 1 1 0 1 0	*4.* 1 1 0 1 0	*4.* 1 1 0 1 0	*4.* 1 1 0 1 0
2. 0 1 0 1 1	*3.* 0 1 0 1 1	*5.* 0 0 1 0 1	*6.* 0 1 1 1 1
* 1 0 1 *	* 1 0 1 *	* * * * *	* 1 * 1 *

Kokkulangevaid elemente (A) on järgnevalt:

objekt i:	2	3	5	6
A:	3	3	0	2

Arvutame uued kaalud:

objekt i:	2	3	5	6
S_{vana}:	1	1	2	0
+ A:	3	3	0	2
S_{uus}:	4	4	2	2

Näeme, et suurima kaaluga objekte on kaks (nr 2 ja nr 3.) Võrdsete kaalude korral tuleb järgmisena elimineerida neist esimene, s.o objekt 2,

Samm 4. Analüüsi on jäänud 4 objekti. Mine Samm 2.

Selguse mõttes läbime ka teise iteratsiooni.

Samm 2. Jne.

Järgnevalt esitame objektide elimineerimise protsessi tabeli kujul.

i \ j	*1*	*2*	*3*	*4*	*5*	S_i	A						Järjestus
1.	1	0	0	0	0	0	–						1.
2.	0	1	0	1	1	0	1	3	–				3.
3.	0	1	0	1	1	0	1	3	5	–			4.
4.	1	1	0	1	0	0	3	–					2.
5.	0	0	1	0	1	0	2	0	2	2	3	–	6.
6.	0	1	1	1	1	0	0	2	4	4	–		5.

i \ j	*1*	*2*	*3*	*4*	*5*	S_i	Kaal					
1.	1	0	0	0	0	0	–					
2.	0	1	0	1	1	0	1	4	–			
3.	0	1	0	1	1	0	1	4	9	–		
4.	1	1	0	1	0	0	3	–				
5.	0	0	1	0	1	0	2	2	4	6	9	–
6.	0	1	1	1	1	0	0	2	6	10	–	

Samm 5. Rakendame nüüd algoritmi samme Samm 1 kuni Samm 4 tunnustele.

Samm 1. Leiame igale väärtusele X_{ij} tema esinemissageduse $Z_i h_j$ reas i.

i \ j	*1*	*2*	*3*	*4*	*5*
1.	1	0	0	0	0
2.	0	1	0	1	1
3.	0	1	0	1	1
4.	1	1	0	1	0
5.	0	0	1	0	1
6.	0	1	1	1	1

0	1
4	1
2	3
2	3
2	3
3	2
1	4

Arvutame tunnustele nende kaalud.

i \ j	*1*	*2*	*3*	*4*	*5*
1.	1	4	4	4	4
2.	2	3	2	3	3
3.	2	3	2	3	3
4.	3	3	2	3	2
5.	3	3	2	3	2
6.	1	4	4	4	4
Sj	12	20	16	20	18

Samm 2. Elimineerime tunnuse 1 kui nõrgima: kaal=12.

Samm 3. Leiame analüüsi jäänud tunnustele kokkulangevate elementide arvu elimineeritava tunnuse suhtes.

1	*2*		*1*	*3*		*1*	*4*		*1*	*5*	
1	0	*	1	0	*	1	0	*	1	0	*
0	1	*	0	0	0	0	1	*	0	1	*
0	1	*	0	0	0	0	1	*	0	1	*
1	1	1	1	0	*	1	1	1	1	0	*
0	0	0	0	1	*	0	0	0	0	1	*
0	1	*	0	1	*	0	1	*	0	1	*
A		2			2			2			0

Arvutame tunnustele uued kaalud:

Tunnus j:	2	3	4	5
S_{vana}:	20	16	20	18
- A:	2	2	2	0
S_{uus}:	18	14	18	18

Samm 4. Analüüsi on jäänud järele 4 tunnust. Mine Samm 2.

Samm 2. Jne.

Järgnevalt esitame algoritmi kogu töö tabeli kujul:

i / j	*1*	*2*	*3*	*4*	*5*
1.	1	0	0	0	0
2.	0	1	0	1	1
3.	0	1	0	1	1
4.	1	1	0	1	0
5.	0	0	1	0	1
6.	0	1	1	1	1
Sj	12	20	16	20	18
	–	18	14	18	18
		16	–	16	14
		12		12	–
		–		6	
				–	
Järjestus	1.	4.	2.	5.	3.

Samm 6. Korrastame andmetabeli read ja veerud.

i \ j	*1*	*3*	*5*	*2*	*4*	Kaal i
1.	1	0	0	0	0	0
4.	1	0	0	1	1	3
2.	0	0	1	1	1	4
3.	0	0	1	1	1	9
6.	0	1	1	1	1	10
5.	0	1	1	0	0	9
Kaal j	0	2	4	8	14	

Võrreldes eelnenud tehnikate tulemustabelitega annab segatehnika tabel hoopis teistsugust informatsiooni, Kuna järjestuse aluseks on sarnasus (kokkulangevuste arv), siis saab jälgida kujunemise dünaamikat lähtudes mingist kindlast objektist (antud näites objekt 1) või tunnusest (tunnus 1). Siin pole eesmärgiks mitte homogeensete elementide gruppide leidmine, vaid nende kujunemise dünaamika nägemine.

Paremaks illustreerimiseks oletagem, et andmetabel kirjeldab konna elutsükli andmeid tema arengus kullesest konnaks jne. Olgu kõik tunnused mõõdetud ei/jah tüüpi skaalal, st kas vastav omadus esineb või ei esine. Tunnus 1 – saba olemasolu, tunnus 3 – tiibade olemasolu, 5 – esijalgade olemasolu, 2 – tagajalgade olemasolu, 4 – toitub putukatest. Seega objekt 1 kirjeldus oleks järgmine: omab saba, pole tiibu, esijalgu ega tagajalgu ning ei toitu putukatest.

Korrastatud tabel toob esile järgmise arengu dünaamika: kullesest kasvab moonde tulemusena sabaga olend, kes ei toitu putukatest (objekt 1), seejärel kasvavad tagajalad (objekt 4), seejärel lisanduvad esijalad ja kaob saba, isend toitub putukatest (objekt 2), ning seejärel toimub mingi moondus, mida antud tabeli tunnused ei mõõda (objekt 3). Seejärel lisanduvad tiivad (objekt 6) ning ühed jalad kaovad ja tekkinud olend ei toitu enam putukatest (objekt 5).

Lisaks võimaldab segatehnika ka objekte ja tunnuseid grupeerida. Selleks peame objektide korral liikuma piki kaalusid ülevalt alla, tunnuste korral vasakult paremale. Kui kaal enam ei kasva, siis algab uute omadustega grupp, st see objekt või tunnus erineb eelmistest rohkem kui temale järgnevates, seega on tekkinud uus kvaliteet:

i \ j	*1*	*3*	*5*	*2*	*4*	Kaal i
1.	1	0	0	0	0	0
4.	1	0	0	1	1	3
2.	0	0	1	1	1	4
3.	0	0	1	1	1	9
6.	0	1	1	1	1	10
5.	0	1	1	0	0	9
Kaal j	0	2	4	8	14	

Selle eeskirja järgi töödeldud tabelist näeme, et objektide osas on leitud 2 gruppi: Go1 (objektid 1, 4, 2, 3 ja 6), Go2 (5), Kui me jälgime objektide kaalu muutust, siis objekti 5 kaal on väiksem talle eelnenud objekti kaalust: 9<10. See tähendab, et pärast objekti 6 on toimunud oluline hüpe – arengus on tekkinud uus kvaliteet, st objekt 5 erineb objektide rühmast objektid 1, 2, 3, 4, 6 rohkem kui sarnaneb neile. Tunnuste osas tekib ainult 1 grupp: Gt1 (tunnused 1, 3, 5, 2 ja 4).

Algoritmi samm-sammulist tööd demonstreerib ka video: segatehnika.

4 Mullati tehnika tuumade eraldamiseks

Eespool kirjeldatud tehnikad olid oma olemuselt väga lihtsad ja interpretatsioonilt kergelt mõistetavad. Kuid andmeanalüüsi aspektist, kui tegemist on suurte andmemassiividega, on niimoodi korrastatud tabelite interpreteerimine küllaltki raske, sest mingis mõttes lähedased objektid ei pruugi sattuda lähestikku. Seetõttu võib väga palju olulist informatsiooni kaotsi minna. Seda esiteks. Teiseks, homogeensete gruppide piiride tõlgendamine on siin subjektiivne, st oleneb paljuski uurija suvast ja tema kogemustest/oskustest näha seaduspärasusi. Tekib küsimus, kas poleks võimalik minimeerida subjektivismi tulemuste interpreteerimisel?

Vajaliku matemaatilise aparatuuri (teooria ja algoritmid) selle tarbeks töötas välja endine TTÜ dotsent Joseph Mullat. Selle järgi on homogeensete elementide grupp määratud kasutatava monotoonse funktsiooniga, st erinevad monotoonsed funktsioonid võivad eraldada erinevad elementide hulgad. Algoritm väljastab homogeense grupi ja elimineerib selle edaspidisest tööst. Seega võib sellese lähenemise tulemusena andmetabeli iga element kuuluda ainult ühte gruppi. Uurija ülesandeks jääb ainult väljastatud gruppide interpreteerimine ja gruppide omavaheline võrdlus.

J. Mullat kirjeldas oma algoritmi ja teooriat artiklite sarjas (1971, 1976, 1977a). Tema algoritmi kirjeldamisel eeldame siin, et me kaalume tabeli X(N,M) elemente X_{ij}.

4.1 J. Mullati algoritm

Samm 1. Määratleda kaalufunktsioon. Valime selleks näiteks $G_{ij} = \pi_X(X_{ij}) = R_{X_{ij},i} * V_{X_{ij},j}$ (elemendi väärtuse esinemissagedus reas korrutatud elemendi esinemissagedusega veerus), $X_{ij} \in X$.

Samm 2. Arvutada igale tabeli elemendile X_{ij} tema kaal G_{ij}. Kui andmetabelis X elemente pole (pärast viimati leitud tuuma elementide elimineerimist lähtetabelist muutus tabel tühjaks), minna LOPP. Leida suurused $L = \min_{X_{ij} \in X} G_{ij}$, $U = \max_{X_{ij} \in X} G_{ij}$.

Samm 3. Arvutada lävekaal U* = L + 0,5(U-L). Käivitada protseduur KIHT(U*) (vt allpool).

Samm 4. Algoritmi Sammul 3 võib tekkida kaks olukorda:

a) protseduuri KIHT(U*) tulemusena kõik hulga X elemendid X_{ij} lülitatakse analüüsist välja;
b) KIHT(U*) ei lülita kõiki elemente X_{ij} analüüsist välja.

Kui käivitub juht a), siis U = U* ja minna Samm 3.

Kui käivitub juht b), siis allesjäänud elementidel $a \in X$ leida vähim kaal inf(U*) ≤ U*. Seejärel rakendada allesjäänud elementidele $a \in X$ protseduuri KIHT(inf(U*)).

Kui KIHT(inf(U*)) rakendamisel tekib olukord a), st et kõik elemendid lülituvad analüüsist välja, siis need elemendid moodustavad **tuuma** lävega inf(U*). Kõrvaldada tuuma elemendid edasisest analüüsist ja minna Samm 2.

Kui KIHT(inf(U*)) rakendamisel tekib olukord b), siis L=inf(U*) ja minna Samm 3.

LÕPP. Kõik tuumad on leitud.

Protseduur KIHT(U*)

```
BEGIN
FOR K=1 TO R DO
    FOR i=1 TO N DO
        FOR j=1 TO M DO
            IF Gij ≤ U* THEN {element Xij lülitada analüüsist välja;
                RXij,i= RXij,i - 1; VXij,j = V Xij,j -1}
            ENDIF
        ENDDO
    ENDDO
ENDDO
END.
```

Protseduuris KIHT(U*) suuruse R väärtus on teadmata, R ≤ N*M. Nagu näeme, põhjustab mingi elemendi väljalülitamine talle vastavate rea- ja veerusageduste vähendamist. See aga omakorda põhjustab temaga seotud elementide kaalude vähenemist – tekivad uued potentsiaalsed kandidaadid väljalülitamiseks, n.ö elemendid, millede kaal muutub sageduste vähenemise tulemusena väiksemaks lävest U*. Kui andmetabel on esimest korda läbitud, siis sellesse alles jäänud elemendid ei pruugi enam omada esialgset kaalu. See aga tähendab, et andmetabeli uuel läbimisel osa alles jäänud elemente võib jälle välja lülituda, mis põhjustab uute elementide kaalu vähenemist. See omakorda põhjustab andmetabeli uue läbimise jne. Kogu see protsess toimub senikaua, kuni
kas

a) andmetabelist on kõik elemendid välja lülitatud,
b) andmetabelisse on jäänud üksteisega seotud elemendid, mille kaal on stabiliseerunud, st need kõik omavad alles jäänud elementide suhtes kaalu ≥ U*.

Eelpool kirjeldatud J. Mullati algoritmi keerukuseks on $O(N^5)$.

Algoritmi iseärasuseks on see, et kui tuum on leitud, siis tema elemendid elimineeritakse edasisest analüüsist ja allesjäänud andmekogumit käsitletakse kui uut lähteandmekogumit. Selline lähenemine tagab algoritmi koonduvuse, st kõik tuumad on leitud, kui lähteandmetabelis peale mingi tuuma leidmist ja selle elementide eemaldamist pole analüüsi jäänud enam ühtegi elementi.

Näide algoritmi töö selgituseks

Algoritmi töö paremaks selgitamiseks vaatleme järgmist näidet. Olgu meil lähteandmetabeliks tabel X(6,5), mida kasutasime andmekorrastusalgoritmide töö selgitamisel.

i \ j	*1*	*2*	*3*	*4*	*5*
1.	1	0	0	0	0
2.	0	1	0	1	1
3.	0	1	0	1	1
4.	1	1	0	1	0
5.	0	0	1	0	1
6.	0	1	1	1	1

Samm 1. Valime kaalufunktsiooniks $G_{ij} = \pi_X(X_{ij}) = R_{X_{ij},i} * V_{X_{ij},j}$ (elemendi väärtuse esinemissagedus reas korrutatud elemendi esinemissagedusega veerus), $X_{ij} \in X$.

Samm 2. Kuna lähteandmetabel pole tühi, arvutame igale tabeli elemendile X_{ij} tema kaalu G_{ij}. Selleks leiame elementide väärtustele sagedusribad nii tabeli ridade kui ka veergude suhtes.

i / j	*1*	*2*	*3*	*4*	*5*
1.	1	0	0	0	0
2.	0	1	0	1	1
3.	0	1	0	1	1
4.	1	1	0	1	0
5.	0	0	1	0	1
6.	0	1	1	1	1

0	1	
4	1	
2	3	
2	3	$R_{X_{ij},i}$
2	3	
3	2	
1	4	

0	4	2	4	2	2
1	2	4	2	4	4

$V_{X_{ij},i}$

Lähtudes leitud sagedustest, oleks elementide kaalude G_{ij} tabel järgmine:

i / j	*1*	*2*	*3*	*4*	*5*
1.	2	8	16	8	8
2.	8	12	8	12	12
3.	8	12	8	12	12
4.	6	12	8	12	4
5.	12	6	4	6	8
6.	4	16	8	16	16

Leiame vähima ja suurima kaalu: L=2, U=16.

Samm 3. Arvutame lävekaalu U* = 2 + 0,5(16-2) = 9. Käivitame protseduuri KIHT(U*=9).
R=1.

I=1. X_{11}=1; G_{11}= 1*2 = 2 < U*=9, elimineerime elemendi X_{11} ja vähendame vastavaid sagedusi R ja V ühe võrra: R(1,1)=1-1=0, V(1,1)=2-1=1;
X_{12}=0; G_{12}= 4*2 = 8 < 9, elimineerime, R(1,0)=4-1=3, V(0,2)=2-1=1;
X_{13}=0; G_{13}= 3*4 =12 (pangem tähele, et tänu eelmise elemendi X13=0 elimineerimisele reasagedus vähenes ühe võrra, mistõttu elemendi X_{14} kaal ka vähenes võrreldes esialgsega (16)!) > 9, jääb analüüsi;
X_{14}=0; G_{14}= 3*2 = 6 < 9, elimineerime, R(1,0)=3-1=2, V(0,4)=2-1=1;
X_{15}=0; G_{15}= 2*2 = 4 < 9, elimineerime, R(1,0)=2-1=1, V(0,5)=2-1=1;

I=2. X_{21}=0; G_{21}= 2*4 = 8 < 9, elimineerime, R(2,0)=2-1=1, V(0,1)=4-1=3;
X_{22}=1; G_{22}= 3*4 =12 > 9, jääb analüüsi;
X_{23}=0; G_{23}= 1*4 = 4 < 9, elimineerime, R(2,0)=1-1=0, V(0,3)=4-1=3;
X_{24}=1; G_{24}= 3*4 =12 > 9, jääb analüüsi;
X_{25}=1; G_{25}= 3*4 =12 > 9, jääb analüüsi;

I=3. X_{31}=0; G_{31}= 2*3 = 6 < 9, elimineerime, R(3,0)=2-1=1, V(0,1)=3-1=2;
X_{32}=1; G_{32}= 3*4 =12 > 9, jääb analüüsi;
X_{33}=0; G_{33}= 1*3 = 3 < 9, elimineerime, R(3,0)=1-1=0, V(0,3)=3-1=2;
X_{34}=1; G_{34}= 3*4 =12 > 9, jääb analüüsi;
X_{35}=1; G_{35}= 3*4 =12 > 9, jääb analüüsi;

I=4. X_{41}=1; G_{41}= 3*1 = 3 < 9, elimineerime, R(4,1)=3-1=2, V(1,1)=1-1=0;
X_{42}=1; G_{42}= 2*4 = 8 < 9, elimineerime, R(4,1)=2-1=1, V(1,2)=4-1=3;
X_{43}=0; G_{43}= 2*2 = 4 < 9, elimineerime, R(4,0)=2-1=1, V(0,3)=2-1=1;
X_{44}=1; G_{44}= 1*4 = 4 < 9, elimineerime, R(4,1)=1-1=0, V(1,4)=4-1=3;
X_{45}=0; G_{45}= 1*1 = 1 < 9, elimineerime, R(4,0)=1-1=0, V(0,5)=1-1=0;

I=5. X_{51}=0; G_{51}= 3*2 = 6 < 9, elimineerime, R(5,0)=3-1=2, V(0,1)=2-1=1;
X_{52}=0; G_{52}= 2*1 = 2 < 9, elimineerime, R(5,0)=2-1=1, V(0,2)=1-1=0;
X_{53}=1; G_{53}= 2*2 = 4 < 9, elimineerime, R(5,1)=2-1=1, V(1,3)=2-1=1;
X_{54}=0; G_{54}= 1*1 = 1 < 9, elimineerime, R(5,0)=1-1=0, V(0,4)=1-1=0;
X_{55}=1; G_{55}= 1*4 = 4 < 9, elimineerime, R(5,1)=1-1=0, V(1,5)=4-1=3;

I=6. X_{61}=0; G_{61}= 1*1 = 1 < 9, elimineerime, R(6,0)=1-1=0, V(0,1)=1-1=0;
X_{62}=1; G_{62}= 4*3 =12 > 9, jääb analüüsi;
X_{63}=1; G_{63}= 4*1 = 4 < 9, elimineerime, R(6,1)=4-1=3, V(1,3)=1-1=0;
X_{64}=1; G_{64}= 3*3 = 9 ≤ 9, elimineerime, R(6,1)=3-1=2, V(1,4)=3-1=2;
X_{65}=1; G_{65}= 2*3 = 6 < 9, elimineerime, R(6,1)=2-1=1, V(1,5)=3-1=2;

Andmetabeli esimese läbimise tulemusena jäid analüüsi järgmised elemendid:

i / j	*1*	*2*	*3*	*4*	*5*
1.	*	*	0	*	*
2.	*	1	*	1	1
3.	*	1	*	1	1
4.	*	*	*	*	*
5.	*	*	*	*	*
6.	*	1	*	*	*

X_{ij}=	0	1
	1	0
	0	3
	0	3
	0	0
	0	0
	0	1

$R_{X_{ij},i}$

0	0	0	1	0	0
1	0	3	0	2	2

X_{ij} $V_{X_{ij},j}$

Nende kaalud selle arvutamise hetkel olid järgmised:

i \ j	*1*	*2*	*3*	*4*	*5*
1.	*	*	12	*	*
2.	*	12	*	12	12
3.	*	12	*	12	12
4.	*	*	*	*	*
5.	*	*	*	*	*
6.	*	12	*	*	*

See, et need kaalud kõikidel elementidel on võrdsed, on juhus. Kui nüüd ekslikult rakendada protseduuri KIHT(inf(U*)), võttes aluseks elementide esialgsed kaalud, arvestamata nendega seotud elementide järgnenud elimineerimist, siis oleks „tuum" leitud, sest inf(U*)=12 ja kõik elemendid lülituksid välja lävekontrolli $G_{ij} \le \inf(U^*)$ tulemusena, kuid tegemist pole tuumaga. Miks – seda selgitame allpool.

Tegelikkuses oleme andmetabeli läbinud ainult üks kord, kusjuures tabelisse allesjäänud elementidega seotud elementide järgneva elimineerimise tulemusena on muutunud need rea- ja veerusagedusd, mille alusel allesjäänud elementide kaalud arvutati. Järgmises tabelis on toodud analüüsi jäänud elementide tegelikud kaalud pärast tabeli esmakordset läbimist (R=1).

i \ j	*1*	*2*	*3*	*4*	*5*
1.	*	*	1	*	*
2.	*	9	*	6	6
3.	*	9	*	6	6
4.	*	*	*	*	*
5.	*	*	*	*	*
6.	*	3	*	*	*

Antud tabel näitab väga ilmekalt, kuivõrd on muutunud elementide kaalud pärast tabeli läbimist võrreldes kaaludega nende arvutamise hetkel. Sellest on selgesti näha, kuidas monotoonsete süsteemide korral määratletakse elementide vahelist seost.

Antud kaalufunktsiooni $G_{ij} = \pi_X(X_{ij}) = R_{X_{ij},i} * V_{X_{ij},j}$ korral on elemendid seotud läbi oma positsiooni rea- ja veerusageduste, kusjuures see seos määratletakse läbi elemendile rakendatava tegevuse. St, kui me elemendi elimineerime, siis me vähendame ka vastavaid sagedusi. Seega, mingi elemendi X_{ij} korral temaga seotud elementide leidmiseks andmetabelist ei pea me füüsiliselt teda võrdlema juba elimineeritud elementidega, et otsustada tema analüüsi jäämise üle. Piisab kui me vähendame sagedusi, mis omakorda toob kaasa elimineeritava elemendiga seotud elementide kaalu muutuse.

R=2. Algseis teiseks iteratsiooniks:

i \ j	*1*	*2*	*3*	*4*	*5*
1.	*	*	0	*	*
2.	*	1	*	1	1
3.	*	1	*	1	1
4.	*	*	*	*	*
5.	*	*	*	*	*
6.	*	1	*	*	*

X_{ij}=	0	1
	1	0
	0	3
	0	3
	0	0
	0	0
	0	1

$R_{X_{ij},i}$

0	0	0	1	0	0
1	0	3	0	2	2

X_{ij} $V_{X_{ij},j}$

I=1. X_{13}=0; G_{13}= 1*1 = 1 < 9, elimineerime, R(1,0)=1-1=0, V(0,3)=1-1=0;

I=2. X_{22}=1; G_{22}= 3*3 = 9 ≤ 9, elimineerime, R(2,1)=3-1=2, V(1,2)=3-1=2;
X_{24}=1; G_{24}= 2*2 = 4 < 9, elimineerime, R(2,1)=2-1=1, V(1,4)=2-1=1;
X_{25}=1; G_{25}= 1*2 = 2 < 9, elimineerime, R(2,1)=1-1=0, V(1,5)=2-1=1;

I=3. X_{32}=1; G_{32}= 3*2 = 6 < 9, elimineerime, R(3,1)=3-1=2, V(1,2)=2-1=1;
X_{34}=1; G_{34}= 2*1 = 2 < 9, elimineerime, R(3,1)=2-1=1, V(1,4)=1-1=0;
X_{35}=1; G_{35}= 1*1 = 1 < 9, elimineerime, R(3,1)=1-1=0, V(1,5)=1-1=0;

I=6. X_{62}=1; G_{62}= 1*1 = 1 < 9, elimineerime, R(6,1)=1-1=0, V(1,2)=1-1=0.

Samm 4. Andmetabeli teistkordsel läbimisel ei jää tabelisse alles ühtegi elementi.
Järelikult oli esialgne lävi U*=9 liiga kõrge. U=U* ning läheme Samm 3.

Samm 3. U*= 2+0,5*(9-2)=5,5. Käivitame protseduuri KIHT(5,5).

i \ j	*1*	*2*	*3*	*4*	*5*
1.	1	0	0	0	0
2.	0	1	0	1	1
3.	0	1	0	1	1
4.	1	1	0	1	0
5.	0	0	1	0	1
6.	0	1	1	1	1

X_{ij}=	0	1
	4	1
	2	3
	2	3
	2	3
	3	2
	1	4

$R_{X_{ij},i}$

X_{ij}	$V_{X_{ij},j}$				
0	4	2	4	2	2
1	2	4	2	4	4

R=1.

I=1. X_{11}=1; G_{11}= 1*2 = 2 < U*=5,5, elimineerime elemendi X11 ja vähendame vastavaid sagedusi R ja V ühe võrra: R(1,1)=1-1=0, V(1,1)=2-1=1;
X_{12}=0; G_{12}= 4*2 = 8 > 5,5, jääb analüüsi;
X_{13}=0; G_{13}= 4*4 =16 > 5,5, jääb analüüsi;
X_{14}=0; G_{14}= 4*2 = 8 > 5,5, jääb analüüsi;
X_{15}=0; G_{15}= 4*2 = 8 > 5,5, jääb analüüsi;

I=2. X_{21}=0; G_{21}= 2*4 = 8 > 5,5, jääb analüüsi;
X_{22}=1; G_{22}= 3*4 =12 > 5,5, jääb analüüsi;
X_{23}=0; G_{23}= 2*4 = 8 > 5,5, jääb analüüsi;
X_{24}=1; G_{24}= 3*4 =12 > 5,5, jääb analüüsi;
X_{25}=1; G_{25}= 3*4 =12 > 5,5, jääb analüüsi;

I=3. X_{31}=0; G_{31}= 2*4 = 8 > 5,5, jääb analüüsi;
X_{32}=1; G_{32}= 3*4 =12 > 5,5, jääb analüüsi;
X_{33}=0; G_{33}= 2*4 = 8 > 5,5, jääb analüüsi;
X_{34}=1; G_{34}= 3*4 =12 > 5,5, jääb analüüsi;
X_{35}=1; G_{35}= 3*4 =12 > 5,5, jääb analüüsi;

I=4. X_{41}=1; G_{41}= 3*1 = 3 < 5,5, elimineerime, R(4,1)=3-1=2, V(1,1)=1-1=0;
X_{42}=1; G_{42}= 2*4 = 8 > 5,5, jääb analüüsi;
X_{43}=0; G_{43}= 2*4 = 8 > 5,5, jääb analüüsi;
X_{44}=1; G_{44}= 2*4 = 8 > 5,5, jääb analüüsi;
X_{45}=0; G_{45}= 2*2 = 4 < 5,5, elimineerime, R(4,0)=2-1=1, V(0,5)=2-1=1;

I=5. X_{51}=0; G_{51}= 3*4 = 12 > 5,5, jääb analüüsi;
X_{52}=0; G_{52}= 3*2 = 6 > 5,5, jääb analüüsi;
X_{53}=1; G_{53}= 2*2 = 4 < 5,5, elimineerime, R(5,1)=2-1=1, V(1,3)=2-1=1;
X_{54}=0; G_{54}= 3*2 = 6 > 5,5, jääb analüüsi;
X_{55}=1; G_{55}= 1*4 = 4 < 5,5, elimineerime, R(5,1)=1-1=0; V(1,5)=4-1=3;

I=6. X_{61}=0; G_{61}= 1*4 = 4 < 5,5, elimineerime, R(6,0)=1-1=0, V(0,1)=4-1=3;
X_{62}=1; G_{62}= 4*4 =16 > 5,5, jääb analüüsi;
X_{63}=1; G_{63}= 4*1 = 4 < 5,5, elimineerime, R(6,1)=4-1=3, V(1,3)=1-1=0;
X_{64}=1; G_{64}= 3*4 = 12 > 5,5, jääb analüüsi;
X_{65}=1; G_{65}= 3*4 = 12 > 5,5, jääb analüüsi.

Andmetabeli esimese läbimise tulemusena jäid analüüsi järgmised elemendid:

i \ j	*1*	*2*	*3*	*4*	*5*
1.	*	0	0	0	0
2.	0	1	0	1	1
3.	0	1	0	1	1
4.	*	1	0	1	*
5.	0	0	*	0	*
6.	*	1	*	1	1

X_{ij}=	0	1
	4	0
	2	3
	2	3
	1	2
	3	0
	0	3

$R_{X_{ij},i}$

0	3	2	4	2	1
1	0	4	0	4	3
X_{ij}		$V_{X_{ij},j}$			

Lähtudes elimineerimisest tingitud sageduste muutusest oleks elementide kaalude G_{ij} tabel enne teistkordset läbimist järgmine:

i \ j	*1*	*2*	*3*	*4*	*5*
1.	*	8	16	8	4
2.	6	12	8	12	9
3.	6	12	8	12	9
4.	*	8	4	8	*
5.	9	6	*	6	*
6.	*	12	*	12	9

R=2. Teisel iteratsioonil langeb välja kaks elementi:

I=1. X15=0, G15=4*1=4 < 5,5, elimineerime, R(1,0)=4-1=3, V(0,5)=1-1=0;

I=4. X43=0, G43=1*4=4 < 5,5, elimineerime, R(4,0)=1-1=0, V(0,3)=4-1=3.

Andmetabeli teise läbimise tulemusena jäid analüüsi järgmised elemendid:

i \ j	*1*	*2*	*3*	*4*	*5*
1.	*	0	0	0	*
2.	0	1	0	1	1
3.	0	1	0	1	1
4.	*	1	*	1	*
5.	0	0	*	0	*
6.	*	1	*	1	1

X_{ij}=	0	1
	3	0
	2	3
	2	3
	0	2
	3	0
	0	3

$R_{X_{ij},i}$

0	3	2	3	2	0
1	0	4	0	4	3
X_{ij}		$V_{X_{ij},j}$			

Lähtudes elimineerimisest tingitud sageduste muutusest oleks elementide kaalude G_{ij} tabel enne kolmandat läbimist järgmine:

i \ j	*1*	*2*	*3*	*4*	*5*
1.	*	6	9	6	*
2.	6	12	6	12	9
3.	6	12	6	12	9
4.	*	8	*	8	*
5.	9	6	*	6	*
6.	*	12	*	12	9

R=3. Kolmandal iteratsioonil ei lange analüüsist välja ühtegi elementi (kontrollige!). St oleme leidnud stabiilse elementide alamhulga.

i \ j	*1*	*2*	*3*	*4*	*5*
1.	*	0	0	0	*
2.	0	1	0	1	1
3.	0	1	0	1	1
4.	*	1	*	1	*
5.	0	0	*	0	*
6.	*	1	*	1	1

X_{ij}=	0	1
	3	0
	2	3
	2	3
	0	2
	3	0
	0	3

$R_{X_{ij},i}$

0	3	2	3	2	0
1	0	4	0	4	3
X_{ij}		$V_{X_{ij},j}$			

Samm 4. Nüüd peame kontrollima, kas tegu on tuumaga. Selleks leiame inf(U*)=minG_{ij}=6 ja käivitame KIHT(inf(U*)=6).

R=1.

I=1. X_{12}=0; G_{12}= 3*2 = 6 ≤ 6, elimineerime, R(1,0)=3-1=2, V(0,2)=2-1=1;
X_{13}=0; G_{13}= 2*3 = 6 ≤ 6, elimineerime, R(1,0)=2-1=1, V(0,3)=3-1=2;
X_{14}=0; G_{14}= 1*2 = 2 < 6, elimineerime, R(1,0)=1-1=0, V(0,4)=2-1=1;

I=2. X_{21}=0; G_{21}= 2*3 = 6 ≤ 6, elimineerime, R(2,0)=2-1=1, V(0,1)=3-1=2;
X_{22}=1; G_{22}= 3*4 =12 > 6, jääb analüüsi;
X_{23}=0; G_{23}= 1*2 = 2 < 6, elimineerime, R(2,0)=1-1=0, V(0,3)=2-1=1;
X_{24}=1; G_{24}= 3*4 =12 > 6, jääb analüüsi;
X_{25}=1; G_{25}= 3*3 = 9 > 6, jääb analüüsi;

I=3. X_{31}=0; G_{31}= 2*2 = 4 < 6, elimineerime, R(3,0)=2-1=1, V(0,1)=2-1=1;
X_{32}=1; G_{32}= 3*4 =12 > 6, jääb analüüsi;
X_{33}=0; G_{33}= 1*1 = 1 < 6, elimineerime, R(3,0)=1-1=0, V(0,3)=1-1=0;
X_{34}=1; G_{34}= 3*4 =12 > 6, jääb analüüsi;
X_{35}=1; G_{35}= 3*3 = 9 > 6, jääb analüüsi;

I=4. X_{42}=1; G_{42}= 2*4 = 8 > 6, jääb analüüsi;
X_{44}=1; G_{44}= 2*4 = 8 > 6, jääb analüüsi;

I=5. X_{51}=0; G_{51}= 3*1 = 3 < 6, elimineerime, R(5,0)=3-1=2, V(0,1)=1-1=0;
X_{52}=0; G_{52}= 2*1 = 2 < 6, elimineerime, R(5,0)=2-1=1, V(0,2)=1-1=0;
X_{54}=0; G_{54}= 1*1 = 1 < 6, elimineerime, R(5,0)=1-1=0, V(0,4)=1-1=0;

I=6. X_{62}=1; G_{62}= 3*4 =12 > 6, jääb analüüsi;
X_{64}=1; G_{64}= 3*4 =12 > 6, jääb analüüsi;
X_{65}=1; G_{65}= 3*3 = 9 > 6, jääb analüüsi.

Andmetabeli esimese läbimise tulemusena jäid analüüsi järgmised elemendid:

i \ j	*1*	*2*	*3*	*4*	*5*
1.	*	*	*	*	*
2.	*	1	*	1	1
3.	*	1	*	1	1
4.	*	1	*	1	*
5.	*	*	*	*	*
6.	*	1	*	1	1
	0	0	0	0	0
	1	0	4	0	4	3
	X_{ij}			$V_{X_{ij},j}$		

X_{ij}=	0	1
	0	0
	0	3
	0	3
	0	2
	0	0
	0	3

$R_{X_{ij},i}$

Nad omavad järgmisi kaalusid:

i \ j	*1*	*2*	*3*	*4*	*5*
1.	*	*	*	*	*
2.	*	12	*	12	9
3.	*	12	*	12	9
4.	*	8	*	8	*
5.	*	*	*	*	*
6.	*	12	*	12	9

R=2. Ette rutates võime öelda, et sellel iteratsioonil ei lange välja ühtegi elementi (kontrollige!), mis tähendab, et oleme leidnud stabiilse elementide alamhulga. Kuna see oli tuuma eksisteerimise kontroll ja analüüsi jäid mõned elemendid, siis antud elementide alamhulk pole tuum. Samal ajal tähendab asjaolu, et osa elemente jäi analüüsi, seda, et lävi U*=6 on liiga madal. L=U*=6 ja läheme Samm 3. Pangem tähele, et me ei lähe analüüsima lähtetabeli elemente, vaid jätkame analüüsi tuumakontrollist allesjäänud elementide alamhulgal. Selline käik on täiesti põhjendatud, sest toimus lävesageduse tõus. Edasisest analüüsist langevad järelikult kindlasti välja ka kõik need elemendid, mis langesid välja eelmise läve (U*=5,5) korral. Seega ei pea me nende kaalusid uuesti arvutama.

Samm 3. U=9, L=6. Arvutame uue läve U* = 6 - 0,5(9 - 6) = 7,5. Rakendame protseduuri KIHT(U*=7,5) järgmisele elementide alamhulgale:

i \ j	*1*	*2*	*3*	*4*	*5*
1.	*	*	*	*	*
2.	*	1	*	1	1
3.	*	1	*	1	1
4.	*	1	*	1	*
5.	*	*	*	*	*
6.	*	1	*	1	1
	0	0	0	0	0
	1	0	4	0	4	3
	X_{ij}			$V_{X_{ij},j}$		

X_{ij}=	0	1
	0	0
	0	3
	0	3
	0	2
	0	0
	0	3

$R_{X_{ij},i}$

R=1. Kuna U*=7,5, siis sellest alamhulgast ei lange välja ükski element (kontrollige!). Nad omavad järgmisi kaalusid:

i/j	*1*	*2*	*3*	*4*	*5*
1.	*	*	*	*	*
2.	*	12	*	12	9
3.	*	12	*	12	9
4.	*	8	*	8	*
5.	*	*	*	*	*
6.	*	12	*	12	9

Samm 4. Peame rakendama tuumakontrolli Inf(7,5)=8. Käivitame protseduuri KIHT(inf(U*)=8).

R=1.

I=2. X_{22}=1; G_{22}= 3*4 =12 > 8, jääb analüüsi;
X_{24}=1; G_{24}= 3*4 =12 > 8, jääb analüüsi;
X_{25}=1; G_{25}= 3*3 = 9 > 8, jääb analüüsi;

I=3. X_{32}=1; G_{32}= 3*4 =12 > 8, jääb analüüsi;
X_{34}=1; G_{34}= 3*4 =12 > 8, jääb analüüsi;
X_{35}=1; G_{35}= 3*3 = 9 > 8, jääb analüüsi;

I=4. X_{42}=1; G_{42}= 2*4 = 8 ≤ 8, elimineerime, R(4,0)=2-1=1, V(1,2)=4-1=3;
X_{44}=1; G_{44}= 1*4 = 4 < 8, elimineerime, R(4,0)=1-1=0, V(1,4)=4-1=3;

I=6. X_{62}=1; G_{62}= 3*3 = 9 > 8, jääb analüüsi;
X_{64}=1; G_{64}= 3*3 = 9 > 8, jääb analüüsi;
X_{65}=1; G_{65}= 3*3 = 9 > 8, jääb analüüsi.

Pärast esimest iteratsiooni on analüüsi jäänud järgmised elemendid:

i \ j	*1*	*2*	*3*	*4*	*5*
1.	*	*	*	*	*
2.	*	1	*	1	1
3.	*	1	*	1	1
4.	*	*	*	*	*
5.	*	*	*	*	*
6.	*	1	*	1	1

X_{ij}=	0	1
	0	0
	0	3
	0	3
	0	0
	0	0
	0	3

$R_{X_{ij},i}$

0	0	0	0	0	0
1	0	3	0	3	3
X_{ij}		$V_{X_{ij},j}$			

Nad omavad järgmisi kaalusid:

i \ j	*1*	*2*	*3*	*4*	*5*
1.	*	*	*	*	*
2.	*	9	*	9	9
3.	*	9	*	9	9
4.	*	*	*	*	*
5.	*	*	*	*	*
6.	*	9	*	9	9

R=2. Sellel iteratsioonil ei lange analüüsist välja ühtegi elementi. Oleme leidnud stabiilse elementide alamhulga. Kuna leidis aset tuumakontroll ning kõik elemendid ei langenud analüüsist välja, pole tegemist tuumaga. Seega on praegune lävi liiga madal L=inf(7,5)=8 ja läheme Samm 3.

Samm 3. L=inf(U*)=8, U=9. Arvutame uue läve: U*=8+0,5(9-8)=8,5. Rakendame protseduuri KIHT(U*=8,5) alamhulgale:

i \ j	*1*	*2*	*3*	*4*	*5*
1.	*	*	*	*	*
2.	*	1	*	1	1
3.	*	1	*	1	1
4.	*	*	*	*	*
5.	*	*	*	*	*
6.	*	1	*	1	1

X_{ij}=	0	1
	0	0
	0	3
	0	3
	0	0
	0	0
	0	3

$R_{X_{ij},i}$

0	0	0	0	0	0
1	0	3	0	3	3
X_{ij}		$V_{X_{ij},j}$			

Nad omavad järgmisi kaalusid:

i \ j	*1*	*2*	*3*	*4*	*5*
1.	*	*	*	*	*
2.	*	9	*	9	9
3.	*	9	*	9	9
4.	*	*	*	*	*
5.	*	*	*	*	*
6.	*	9	*	9	9

R=1. Kaalude tabelist on näha, et ükski element analüüsist välja ei lange. St esimesel iteratsioonil oleme leidnud stabiilse elementide alamhulga. Rakendame sellele tuumakontrolli.

Samm 4. Inf(U*=8,5)=9. Rakendame protseduuri KIHT(inf(U*)=9).
R=1.

I=2. X_{22}=1; G_{22}= 3*3 = 9 $\leq$ 9, elimineerime, R(2,1)=3-1=2, V(1,2)=3-1=2;
X_{24}=1; G_{24}= 2*3 = 6 < 9, elimineerime, R(2,1)=2-1=1, V(1,4)=3-1=2;
X_{25}=1; G_{25}= 1*3 = 3 < 9, elimineerime, R(2,1)=1-1=0, V(1,5)=3-1=2;

I=3. X_{32}=1; G_{32}= 3*2 = 6 < 9, elimineerime, R(3,1)=3-1=2, V(1,2)=2-1=1;
X_{34}=1; G_{34}= 2*2 = 4 < 9, elimineerime, R(3,1)=2-1=1, V(1,4)=2-1=1;
X_{35}=1; G_{35}= 1*2 = 2 < 9, elimineerime, R(3,1)=1-1=0, V(1,5)=2-1=1;

I=6. X_{62}=1; G_{62}= 3*1 = 3 < 9, elimineerime, R(6,1)=3-1=2, V(1,2)=1-1=0;
X_{64}=1; G_{64}= 2*1 = 2 < 9, elimineerime, R(6,1)=2-1=1, V(1,4)=1-1=0;
X_{65}=1; G_{65}= 1*1 = 1 < 9, elimineerime, R(6,1)=1-1=0, V(1,5)=1-1=0.

KIHT(inf(U*)) rakendamise tulemusena lülitati kõik elemendid analüüsist välja – eraldatud elementide alamhulk on tuum kvaliteediga U*=9:

i \ j	*1*	*2*	*3*	*4*	*5*
1.	*	*	*	*	*
2.	*	1	*	1	1
3.	*	1	*	1	1
4.	*	*	*	*	*
5.	*	*	*	*	*
6.	*	1	*	1	1

Vastavalt algoritmile toimub nüüd tuuma elementide elimineerimine lähteandmetabelist ja tuleb minna Samm 2.

Samm 2. Saadud andmetabel on lähteks järgmise tuuma leidmisel:

i \ j	*1*	*2*	*3*	*4*	*5*
1.	1	0	0	0	0
2.	0	*	0	*	*
3.	0	*	0	*	*
4.	1	1	0	1	0
5.	0	0	1	0	1
6.	0	*	1	*	*
0	4	2	4	2	2
1	2	1	2	1	1
X_{ij}			$V_{X_{ij},j}$		

X_{ij}=	0	1
	4	1
	2	0
	2	0
	2	3
	3	2
	1	1

$R_{X_{ij},i}$

Antud kohal me lõpetame oma näite. Lugeja võib nüüd ise jätkata, sest algoritmi kõik sammud on vähemalt ühe korra läbi mängitud.

Algoritmi samm-sammulist tööd demonstreerib ka video: Mullati tuumad.

Kui lähtesüsteemi struktuuri seisukohalt eraldatud tuuma iseloomustada, siis näeme, et tema elimineerimine lähteandmetabelist on põhjustanud selle struktuuri olulist muutust. Nimelt analüüsi jäänud elementide seosed (vt ühtesid) on muutunud, samuti on selginenud nullide struktuur.

4.2 Täiendavad võimalused tuuma mõiste määratlemisel

Ülalpool kirjeldatud Mullati algoritmi rakendamisel võivad tekkida mitmed eri olukorrad, mille lahendamine põhjustab teatud järeleandmisi tuuma mõiste määratlemisel. Sellised olukorrad oleksid järgmised.

1) Toodud näitest nägime, et tuum kujutab endast mingis mõttes sarnaste elementide alamhulka. Vastavalt monotoonsuse omadusele on iga järgneva tuuma kvaliteet – headus, mõõdetuna lävekaaluga inf(u*) – mitte suurem eelmise eraldatud tuuma kvaliteedist. Praktikas see igakord nii ei väljendu. Võimalikud on olukorrad, kus algoritm ei suuda eraldada üksteisest kahte (või enamat) tuuma. Kompromissina väljastatakse nad üheskoos.

2) Võimalikud on olukorrad, kus algoritm võib jääda nn „igavesse tsüklisse". Need on olukorrad, kus tuumal puudub range struktuur – elemendid moodustavad laigu, mitte ristküliku – tuuma elementide arv eri ridades ja veergudes on erinev. Selle olukorra lahendamiseks kasutatakse tavaliselt kahte võimalikku lähenemist:

a) sisuline lähenemine – kui elementide alamhulka kuuluvad ainult ühe väärtusega elemendid, loetakse see tuumaks. Antud olukorras lähtutakse eeldusest, et elementide alamhulk on homogeenne ja seega sisuliselt interpreteeritav;

b) algoritmiline lähenemine – inf(u*)=inf(u*)+1.

3) Võimalikud on juhud, kus tuuma kuulub üks/kaks elementi. Analüüsi seisukohalt ei paku sellised alamhulgad mingit huvi. Seepärast luuakse võimalused piiritleda tuuma elementide arvu, st kasutaja saab defineerida alamhulga minimaalse elementide arvu, mille korral tegemist on veel sisulise tuumaga.

Lisame siia ühe soovituse algoritmi kiiruslike näitajate parandamiseks. Siiani kasutasime läve U* arvutamise valemis koefitsienti 0,5. Algoritmi kiirema koonduvuse huvides soovitatakse kasutada koefitsienti 0,381 (U*=L+0,381(U-L)).

5 Väljavõttude tehnika

Andmetabelite korrastusalgoritmides kasutatav tehnika (+ ja -) objektide/tunnuste ümber järjestamiseks oli imelihtne – elimineeriti üks tabeli rida/veerg ja hinnati, kuivõrd ta mõjutas ülejäänuid. Selle hinnangu järgi otsustati, milline tabeli rida/veerg järgmisena elimineerida.

J. Mullati algoritmis kasutatav tehnika oli keerulisem. Tabeli elemente võidakse elimineerida korraga mitmeid, sõltuvalt lävesagedusest. Seejuures võib tekkida olukord, kus andmetabeli läbimisel osutuvad elimineerituks kõik selle elemendid. Nimetatu on tingitud eelkõige sellest, et järgmiste elementide kaal (mõju) arvutatakse ümber kohe peale mingi elemendi elimineerimist andmetabelist.

Seetõttu selgub lõplik elementide hulk, millele rakendatakse tuuma tingimuste täidetuse kontrolli, alles pärast andmetabeli mitmekordset läbimist. Kui tuum on leitud, siis tema elemendid elimineeritakse andmetabelist ja alles jäänud elementide hulgale rakendatakse jälle ülalpool kirjeldatud tehnikat. See välistab olukorra, kus üks andmetabeli element võib kuuluda mitmesse tuuma (elementide alamhulka).

Järgnevalt kirjeldame tehnikat, mille korral on lubatud elemendi kuulumine mitmesse alamhulka. Antud tehnika on tunduvalt keerulisem eelpool kirjeldatutest, kuna tõsiseks probleemiks on korduvate alamhulkade väljastamise vältimine.

Vastavat tehnikat kirjeldame järgmise probleemiseade kaudu.

Keeleteadlastel on sagedaseks ülesandeks leida tekstis esinevate tähtede esinemissagedused. Selle lahendamiseks võetakse mingi tekstiosa ja loendatakse, missugused tähed mitu korda esinevad. Selliseid tekstiosa väljavõtteid ja loendusi tehakse mitmeid. Saadud andmete alusel arvutatakse iga tähe keskmine esinemissagedus (%) tekstis. Nagu näeme, on ülesanne kergesti lahendatav, kuna tegu on ainult tähtede loendamisega. Lahendatav on see tekstiosa ühekordse läbimisega.

Muudame nüüd ülesande keerukamaks. Oletame, et teadlane peab leidma täheühendite esinemissagedused sõnade alguste suhtes (täpsustus: täheühendi all mõtleme sõnas järjestikku asuvaid tähti). Näiteks olgu antud sõnad MERI ja METS. Püstitatud ülesande lahenduseks oleks siis M=2, ME=2, MER=1, MET=1, MERI=1, METS=1. Nagu näeme, on jälle tegemist loendusülesandega, kuid selle raskusaste on oluliselt suurenenud, sest tekstiosa läbimiste arv on tunduvalt kasvanud. Seejuures on teksti läbimiste arv määratud erinevate täheühendite arvuga sõnade alguse suhtes.

Antud ülesanne võimaldab väga hästi selgitada meie poolt kirjeldatavat tehnikat. Nimelt igale täheühendile vastab terve tekstiosa suhtes mingite sõnade alamhulk. Kuna suvaline täheühend võib sisalduda mingis teises täheühendis, siis meie poolt kirjeldatava tehnika mõttes võib see kuuluda mitmesse erinevasse sõnade alamhulka.

Meie poolt kirjeldatava tehnika esitame järgmise algoritmina (märkus: kõik käesolevas peatükis kirjeldatavad algoritmid esitatakse tõestusteta).

Algoritm S1

Samm 1. Valime algustähe, mida me veel pole vaadelnud. Kui sellist pole, mine LOPP. Eraldame tekstist kõik sõnad, mis algavad selle tähega (moodustame alamhulga e väljavõtu 1). Leitud sõnade arv määrab selle tähe esinemissageduse. Leiame väljavõtus sisalduvate sõnade teisel positsioonil asuvate tähtede esinemissagedused.

Samm 2. Valime ühe neist teise positsiooni tähtedest, mida varem pole vaadeldud, täheks, mille alusel teeme väljavõtu. Kui sellist pole, mine Samm 1. Eraldame kõik sõnad, mis algavad niiviisi moodustatud täheühendiga sõna algusest (väljavõtt 2 ehk väljavõtt väljavõtust 1). See määrab talle vastava täheühendi esinemissageduse alates sõna algusest. Leiame nende kolmandal positsioonil asuvate tähtede esinemissagedused.

Samm i. Valime ühe neist i-nda positsiooni tähtedest, mida varem pole vaadeldud, täheks, mille alusel teeme väljavõtu. Kui sellist pole, mine Sammule (i-1). Eraldame kõik sõnad, mis algavad selliselt moodustatud täheühendiga sõna algusest – väljavõtt väljavõtust (i-1). See määrab talle vastava täheühendi esinemissageduse alates sõna algusest. Leiame väljavõtus sisalduvate sõnade (i+1)-ndal positsioonil asuvate tähtede esinemissagedused.

LÕPP. Kõik täheühendid on leitud.

Selle algoritmi suurim sammude arv mingi täheühendi leidmisel on määratud pikima sõna tähtede arvuga.

Selgitame algoritmi tööd järgmise näite kaudu.

Oletame, et tekstiosa koosneb sõnadest

MERI METS MIISU MERSU RIST RISU

Ülesanne

Leida tekstiosas sisalduvate täheühendite esinemissagedused sõnade alguse suhtes.

Lahendus

Samm 1. M=4. Väljavõtt: MERI METS MIISU MERSU
Samm 2. ME=3. Väljavõtt: MERI METS MERSU
Samm 3. MER=2. Väljavõtt: MERI MERSU
Samm 4. MERI=1.
Samm 4. MERS=1. Väljavõtt: MERSU
Samm 5. MERSU=1
Samm 3. MET=1. Väljavõtt: METS
Samm 4. METS=1
Samm 2. MI=1. Väljavõtt: MIISU
Samm 3. MII=1. Väljavõtt: MIISU
Samm 4. MIIS=1. Väljavõtt: MIISU
Samm 5. MIISU=1
Samm 1. R=2. Väljavõtt: RIST RISU
Samm 2. RI=2. Väljavõtt: RIST RISU
Samm 3. RIS=2. Väljavõtt: RIST RISU
Samm 4. RIST=1
Samm 4. RISU=1
LOPP. Kõik täheühendid sõnade alguse suhtes leitud.

Eelpool püstitatud ülesannet saab lahendada ka teise, eelmisele väga sarnase, algoritmi alusel.

Algoritm S2

Samm 0. Leiame sõnade algustähtede esinemissagedused. Väljasta ühetäheliste sõnaühendite esinemissagedused.
Samm 1. Valime sõnade algustähe (esimese positsiooni tähe), mis pole veel vaatluse all olnud. Kui selliseid pole, mine LOPP. Eraldame tekstist kõik sõnad, mis algavad selle tähega (teeme väljavõtu 1). Leiame väljavõtus sisalduvate sõnade teisel positsioonil asuvate tähtede esinemissagedused. Need kirjeldavad neile vastavad kahetähelised täheühendid sõnade alguse suhtes (esimeseks täheks on algustäht, mille alusel valiti sõnad väljavõttu).
Samm 2. Valime ühe neist teise positsiooni tähtedest, mis pole veel selles väljavõtus vaatluse all olnud, täheks, mille alusel teeme väljavõtu 2. Kui selliseid pole, mine Samm 1. Eraldame kõik sõnad, mis algavad selliselt moodustatud täheühendiga – esimese ja teise positsiooni täht – teine väljavõtt e väljavõtt väljavõtust 1. Leiame nende kolmandal positsioonil asuvate tähtede esinemissagedused, mis kirjeldavad neile vastavad kolmetähelised täheühendid sõnade alguse suhtes.
Samm i. Valime ühe neist i-nda positsiooni tähtedest, mis pole veel selles väljavõtus vaatluse all olnud, täheks, mille alusel teeme väljavõtu (i+1). Kui selliseid pole, mine Samm (i-1). Eraldame kõik sõnad, mis algavad selliselt moodustatud täheühendiga sõna algusest (väljavõtt väljavõtust i). Leiame väljavõtus sisalduvate sõnade (i+1)-sel positsioonil asuvate tähtede esinemissagedused, mis kirjeldavad neile vastavad (i+1)-sed täheühendid sõnade alguse suhtes.
LOPP. Kõik täheühendid sõnade alguse suhtes leitud.

Nii nagu algoritmi S1 korral, on ka siin suurim sammude arv täheühendi leidmisel määratud tähtede arvuga pikimas sõnas.

Rakendades algoritmi S2 eelmisele näitele
MERI METS MIISU MERSU RIST RISU,
saame tulemuseks:
Samm 0. M=4, R=2. Väljavõtt: MERI METS MIISU MERSU
Samm 1. ME=3, MI=1. Väljavõtt: MERI METS MERSU
Samm 2. MER=2, MET=1. Väljavõtt: MERI MERSU
Samm 3. MERI=1, MERS=1. Väljavõtt: MERSU
Samm 4. MERSU=1
Samm 3. METS=1
Samm 2. MII=1. Väljavõtt: MIISU
Samm 3. MIIS=1. Väljavõtt: MIISU
Samm 4. MIISU=1

Samm 1. RI=2. Väljavõtt: RIST RISU
Samm 2. RIS=2. Väljavõtt: RIST RISU
Samm 3. RIST=1, RISU=1
LOPP. Kõik täheühendid sõnade alguse suhtes on leitud.

Nagu näeme, erineb saadud tulemus algoritmi S1 omast ainult täheühendite väljastamise järjekorra poolest. Algoritmide keerukus on ühesugune – mõlemad läbivad töö käigus n-puu, kus n on mingist tipust väljuvate harude arv.

Seejuures algoritmi S1 tulemus annab meile rohkem informatsiooni, kuna võimaldab väljastada täheühendid korrastatuna, st tähestikulises järjekorras. Algoritm S2 seda ei võimalda.

Üheks elementaartegevuseks mõlemas algoritmis on tagasipöördumine (*backtracking*), mis käivitub, kui selgub, et analüüsitavat täheühendit ei saa rohkem laiendada, kuna tekstis pole rohkem seda täheühendit sisaldavaid sõnu.

Muudame nüüd täheühendite leidmise ülesande natuke keerulisemaks ja näitame, et ülalpool kirjeldatud tehnika võimaldab väga efektiivselt, ilma mingite täiendusteta, lahendada ka oluliselt keerukamaid ülesandeid kui senini lahendasime.

Ülesanne

Leida tekstiosas sisalduvate kõikide täheühendite esinemissagedused sõltumata nende asukohast sõnas.

Selgitame seda järgmise näite kaudu.
Oletame, et tekstiosa koosneb sõnadest
ERIMERI ERIM MERI METS MERSU

Lahendus

Kasutame ülesande lahendamiseks algoritmi S1, asendades selles Samm 1-s termini „sõna” terminiga „sõnaosa”. Uue Samm 1 sõnastus oleks siis järgmine:

Samm 1. Valime algustähe, mida me pole veel vaadelnud. Kui sellist pole, mine LOPP. Eraldame tekstist kõik sõna**o**sad, mis algavad selle tähega, antud tähest sõna lõpuni (teeme väljavõtu 1). Leitud sõna**osade** arv määrab selle tähe esinemissageduse. Leiame väljavõtus sisalduvate sõna**osade** teisel positsioonil asuvate tähtede esinemissagedused.

Algoritmi rakendus, valides esimeseks algustäheks „E”, annaks meie näite korral järgmise tulemuse.
Samm 1. E=6. Väljavõtt oleks järgmine: ERIMERI ERI ERIM ERI ETS ERSU
Samm 2. ER=5. Väljavõtt: ERIMERI ERI ERIM ERI ERSU
Samm 3. ERI=5 Väljavõtt: ERIMERI ERI ERIM ERI ERSU
Samm 4. ERIM=2. Väljavõtt: ERIMERI ERIM
Samm 5. ERIME=1. Väljavõtt: ERIMERI
Samm 6. ERIMER=1
Samm 7. ERIMERI=1
Samm 3. ERS=1. Väljavõtt: ERSU
Samm 4. ERSU=1
Samm 2. ET=1. Väljavõtt: ETS
Samm 3. ETS=1
Samm 1. M=5. Väljavõtt: MERI M MERI METS MERSU
Samm 2. ME=4. Väljavõtt: MERI MERI METS MERSU
Samm 3. MER=3. Väljavõtt: MERI MERI MERSU
Samm 4. MERI=2. Väljavõtt: MERI MERI
Samm 4. MERS=1. Väljavõtt: MERSU
Samm 5. MERSU=1
Samm 3. MET=1. Väljavõtt: METS
Samm 4. METS=1
Samm 1. R=5. Väljavõtt: RIMERI RI RIM RI RSU

Kes on mõistnud algoritmi loogikat, sellele ei tee mingit raskust ise jätkata.

Nagu näeme, ei ole kasutatud tehnika muutunud, kuid muutunud on ainult lähteandmehulga formeerimise tingimused. Seejuures oleme viimase ülesande raames lahendanud ka esialgse ülesande – leida üksikute tähtede esinemissagedused tekstis.

Alapeatüki kokkuvõtteks

Käesolevas alapeatükis kirjeldatud algoritmid S1 ja S2 erinevad klassikalistest algoritmidest eelkõige selle poolest, et nendesse on orgaaniliselt sulatatud nn väljavõttude tehnika. Selle asemel, et otsida teatud omadustega sõnu tervest tekstiosast, kasutatakse siin suunatud liikumist ehk nn väljavõttu väljavõtust. Selle tulemusena toimub sobivate sõnade otsimine ainult eelmise täheühendi eraldamiseks olnud sõnade seast, mis garanteerib, et vaatluse all on kõik nõutud omadustega sõnad vaadeldavast tekstiosast. Samas on tegu ka monotoonse käsitlusega, sest sõnade esinemissagedus väljavõttude ahelas saab ainult väheneda.

Iga eraldatav täheühend on määratud ta esinemissagedusega. Antud juhul esinemissagedus on funktsiooniks, mille alusel otsustatakse algoritmi mingi sammu lõpetamise ja *backtrackingu* kohta. Tegemist on monotoonse funktsiooniga, kuna täheühendi laiendamisel mingi tähe- või täheühendiga tema esinemissagedus võib ainult mitte kasvada, st, kas ainult samaks jääda või siis väheneda.

Järgmistes peatükkides käsitleme keerulisemaid meetodeid, mis on tervenisti arendatud endises TTÜ informaatikainstituudis.

Hüpoteeside generaator

Järgnevalt siseneme andmekaeve valdkonda ja käsitleme meetodit „Hüpoteeside generaator”.

6 Andmete klasterdamine

Selles jaotises käsitleme eelmises peatükis kirjeldatud väljavõttude tehnikat andmetabelist klastrite leidmiseks. Eesmärgiks on ette antud andmestu sisemise struktuuri avamine.

Määratleme klastri kui ühesuguse käitumisega elementide rühma. Meie eesmärgiks on leida range struktuuriga klastreid, st kõiki klastrisse kuuluvad objekte kirjeldatakse ühesuguste tunnuste kaudu, kus tunnuste arv leitud klastris K ≤ M, kus M on tunnuste arv töödeldavas andmetabelis.

Seda on lihtne selgitada hulgateoreetilise mõiste „lõige” abil .

Olgu antud 2 hulka X1 ja X2.

Definitsioon 1. Hulkade X1 ja X2 lõikeks X1 ∩ X2 nimetatakse elementide hulka {Xij}, Xij ∈ X1, X2.

Näide 1

X1 = {a,b,c} X2 = {b,d}

X1 ∩ X2 = {b}

Olgu antud andmetabel X(N,M) = {Xi}, i=1,…,N, Xi = {Xij}, j=1,…,M; Xij=hj=0,1,…,Kj-1.

Definitsioon 2. Lõikeks tabelil X = {Xi} nimetatakse tunnuste {j} väärtuste hulka {Xij = hj}, mis on positsiooniliselt ühised kõikidele objektidele {Xij: ∀i, Xij ∈ Xi}

Näide 2

Olgu antud andmetabel X(2,3).

i \ j	*1*	*2*	*3*
1.	1	2	2
2.	1	1	2
Lõige	1	*	2

$\cap X = {}_i \cap X_i$ = {1 * 2}, kus * tähistab tühja elementi.

Kui i=1, siis ∩ X = X,

Kui i=0, siis ∩ X = Ø.

Oluline on ka teadmine, et kui iga tunnuse j korral Kj=K, siis max $N=K^M$ ning kõikvõimalikke lõikeid $(K+1)^M$-1. Järgnevalt vaatame, kuidas klassikaliselt lõigete leidmise ülesannet lahendatakse.

Algoritm A1

Leitakse lõiked üle objektipaaride, seejärel üle objektikolmikute, -nelikute, …, -M-ikute. Sellise lähenemise korral on 2^N-1 võimalust lõikamiseks.

Näide

Algtabel

X	T1	T2	T3
O1	1	2	2
O2	1	1	2
O3	1	3	1
O4	1	1	2

Objektide lõikamine.

Korduvad lõiked on värvitud kollaseks.

O1	1	2	2	O1	1	2	2	O1	1	2	2	O2	1	1	2	O2	1	1	2	O3	1	3	1
O2	1	1	2	O3	1	3	1	O4	1	1	2	O3	1	3	1	O4	1	1	2	O4	1	1	2
Tulem	1	*	2		1	*	*		1	*	2		1	*	*		1	1	2		1	*	*

O1	1	2	2	O1	1	2	2	O1	1	2	2	O2	1	1	2
O2	1	1	2	O2	1	1	2	O3	1	3	1	O3	1	3	1
O3	1	3	1	O4	1	1	2	O4	1	1	2	O4	1	1	2
Tulem	1	*	*		1	*	2		1	*	*		1	*	*

O1	1	2	2
O2	1	1	2
O3	1	3	1
O4	1	1	2
Tulem	1	*	*

Näeme, et leitakse palju korduvaid lõikeid. Tühilõiked on antud näites välistatud tingituna tabeli olemusest – kõigil objektidel on tunnuse T1 väärtus ühesugune, st T1=1.

Algoritm A2

Leitakse lõiked üle kõikide objektipaaride, seejärel lõiked üle kõikide saadud lõigete paaride (objektipaaride lõigete), seejärel lõiked üle niiviisi saadud lõigete paaride (objektipaaride lõigete paaride lõigete) jne, jättes kogu aeg järele ainult originaalsed lõiked, kuni ühtegi uut lõiget enam ei lisandu või kuni kõik saadud lõiked on tühilõiked.
Sellise lähenemise korral max iteratsioonide arv = $\min\{M-1, \log_2(\max_j |h_j|)\}$

Näide

Algtabel

X	T1	T2	T3
O1	1	2	2
O2	1	1	2
O3	1	3	1
O4	1	1	2

Objektide lõikamine:

O1	1	2	2	O1	1	2	2	O1	1	2	2	O2	1	1	2	O2	1	1	2	O3	1	3	1
O2	1	1	2	O3	1	3	1	O4	1	1	2	O3	1	3	1	O4	1	1	2	O4	1	1	2
Tulem	1	*	2		1	*	*		1	*	2		1	*	*		1	1	2		1	*	*

Leitud lõigetest (tulem) formeerub uus tabel X1:

X1	T1	T2	T3
L1	1	*	2
L2	1	*	*
L3	1	1	2

Lõigete (tabel X1) lõikamine:

L1	1	*	2	L1	1	*	2	L2	1	*	*
L2	1	*	*	L3	1	1	2	L3	1	1	2
Tulem	1	*	*		1	*	2		1	*	*

Näeme, et lõigete lõikamisel ühtegi uut lõiget ei leitud.

X2	T1	T2	T3

Tühi, uusi lõikeid pole

Taas näeme, et leitakse palju korduvaid lõikeid. Tühilõiked on antud näites välistatud tabeli olemusest tingituna – kõigil objektidel on tunnuse T1 väärtus ühesugune, st T1=1.

Algoritmid A1 ja A2 on väga töömahukad ja seda kahel põhjusel.

1) Objektide lõikamine toimub stiihiliselt, st me ei oska öelda, missuguseid objekte (lõikeid) me peaksime lõikama uute lõigete saamiseks. Sellel põhjusel teeme tohutult tühja tööd, sest suurem osa lõikeid on kas tühjad või korduvad.

2) Väga tülikas on määrata saadud lõike originaalsust – kas lõige on korra juba väljastatud või mitte.

Algoritm A3

```
FOR I1=1 TO N DO
    FOR I2=I1+1 TO N DO
            leia lõige
            IF lõige=∅ THEN NEXT I2
            FOR I3=I2+1 TO N DO
                    leia lõige
                    IF lõige=∅ THEN NEXT I3
                            …
                            FOR IN=I(N-1)+1 TO N DO
                            …
                            NEXT IN
                            …
    NEXT I2
NEXT I1
```

Nagu näeme, toimub igas FOR tsüklis andmetabeli järgmise objekti lõikamine eelmise FORi tulemuslõikega. Juhul, kui tulemuseks on tühilõige, tunnistatakse antud haru tupikuks, st järgnevad FOR tsüklid selle sees on mõttetud, sest edasine lõikamine annaks alati tulemuseks tühja lõike. Sellise kontrollimehhanismi olemasolu võimaldab minimeerida tühilõigete arvu, mistõttu algoritm on kahest eelmisest efektiivsem. Algoritmi A3 korral jääb aga alles teine puudus – leitud lõike korduvus. Kui me kujutaksime lõigete genereerimist hierarhilise puuna, siis algoritmi A3 korral on korduva lõike leidmine samaväärne korduva sattumisega juba läbitud harru. Seejuures ei tea me ka öelda, kas selle haru alamharud on juba läbitud või veel läbimata. Algoritmi A3 korral läheks sellise kontrollimehhanismi loomine väga kalliks, sest lõigete genereerimine toimub stiihiliselt (objekte lõigatakse sellises järjekorras, nagu need paiknevad lähteandmetabelis X). Ka selle algoritmi korral on tegemist rohke ülearuse tööga.

Näide

Algtabel

X	T1	T2	T3
O1	1	2	2
O2	1	1	2
O3	1	3	1
O4	1	1	2

Lõikamine

O1	1	2	2
O2	1	1	2
Tulem	1	*	2
∩			
O3	1	3	1
Tulem	1	*	*
∩			
O4	1	1	2
Tulem	1	*	*

O1	1	2	2
O3	1	3	1
Tulem	1	*	*
∩			
O4	1	1	2
Tulem	1	*	2

O1	1	2	2
O3	1	3	1
Tulem	1	*	*
∩			
O4	1	1	2
Tulem	1	*	*

O1	1	2	2	O2	1	1	2	O2	1	1	2
O4	1	1	2	O3	1	3	1	O4	1	1	2
Tulem	1	*	2	Tulem	1	*	*	Tulem	1	1	2
				∩							
				O4	1	1	2				
				Tulem	1	*	*				

O3	1	3	1
O4	1	1	2
Tulem	1	*	*

Näeme, et kõikide nimetatud algoritmide korral kerkivad üles järgmised probleemid.

1) Kuidas vältida tühilõikeid?
2) Kuidas vältida korduvaid lõikeid?
3) Kuidas muuta lõigete leidmine juhitavaks? St milliseid tehnikaid kasutada andmetabeli läbimisel, et tingimused 1) ja 2) oleksid rahuldatud, st et ei leitaks tühilõikeid ega korduvaid lõikeid.
Eelpool kirjeldatud algoritmide põhiliseks puuduseks on see, et ei teata, milliseid objekte tuleks omavahel lõigata originaalse lõike saamiseks.

Allpool kirjeldame monotoonsete süsteemide lähenemist nimetatud probleemide lahendamisel. Monotoonsete Süsteemide Teooriast lähtuvalt me peaksime

1) leidma monotoonse kaalufunktsiooni ja määratlema sihifunktsiooni, mis saavutaks oma globaalse maksimumi (miinimumi) antud omadustega objektide alamhulgal (st eraldaks lõike),
2) leidma tegevused, mis garanteeriksid süsteemi monotoonsuse ja leitud lõike kordumatuse.

7 Monotoonsete süsteemide lahendus

Lähtekoht: väärtuse esinemissagedus andmetabelis identifitseerib lõike elemendid üheselt.

Näide

Olgu antud andmetabel X(2,3).

i \ j	*1*	*2*	*3*
1.	1	2	2
2.	1	1	2

Arvutame vastava sagedustabeli:

väärtus	*1*	*2*	*3*
1	**2**	1	0
2	0	1	**2**

Sagedustabelist on näha, et kui elemendi Xij sagedus = N (antud näites N=2), siis kuulub element lõikesse. Edasi on tarvis leida sobivad tehnikad, et välistada tühjade ja korduvate lõigete leidmine Sobivaks tehnikaks osutub MS elementaartehnikate peatükis viimasena käsitletud väljavõttude tehnika, mida tuleb täiendada kordusi välistatavate tegevustega.

Algoritmi tööpõhimõtte selgitamine

Järgnevalt lisaksime mõned kommentaarid algoritmi töö selgitamiseks.

Algoritmi tööpõhimõte on lihtne:

1) eraldatakse teatud omadustega objektide alamhulk $X_{t+1} \subset X_t$ (X_0 on algtabel).
2) seejärel leitakse lõige üle selle alamhulga X_{t+1}.
Neid kahte sammu korratakse (t=0,1,...,U, 0<U≤M) seni, kuni leidub veel eraldamata alamhulki $X_{t+1} \subset X_t$, .

Lõike leidmine hulgal X_{t+1} toimub talle vastava sagedustabeli kaudu järgmiselt.

Alamhulga X_{t+1} mingi elemendi maksimaalne esinemissagedus MAX sagedustabelis FT_{t+1} on määratud selle alamhulga eraldamise aluseks olnud elementaarkonjunktsiooni (see moodustub

väljavõttude jada aluseks olnud tunnus.väärtustest, mille alusel eelnevad väljavõtud on tehtud) esinemissagedusega sagedustabelis FT_t. Järelikult kõik X_{t+1} elemendid, mille sagedus võrdub MAX, esinevad üheaegselt kõikides X_{t+1} objektides ja moodustavad seega lõike üle hulga X_{t+1}.

Ühtlasi saame algoritmi tööd suunata nn sageduspiiranguga, st võime ette anda otsitavate lõigete minimaalse esinemissageduse SP (näiteks SP=2. Sel juhul eraldatakse kõik lõiked, mille esinemissagedus on suurem-võrdne 2).

Monotoonsete Süsteemide Teooria seisukohalt lõige on

1) tuum J. Mullati mõttes,

2) $\pi_{X'\setminus\{c\}}(X_{ij}) \leq \pi_{X'}(X_{ij})$, kus X' on suvaline hulga X alamhulk.

Keda huvitab vastav matemaatiline käsitlus teoreemide tasemel, siis on see esitatud artiklis (Kuusik 1993).

Algoritm

Et lõigata sobivaid objektide hulki, valitakse järgmine tipp elemendi sageduse järgi (sagedustabelist) ning korduste vältimiseks kasutatakse kolme elimineerimis-tehnikat. Tegemist on sügavuti otsinguga.

Iga objektidest tehtud väljavõtu jaoks leitakse sellele vastav sagedustabel. Null esialgses sagedustabelis tähendab, et vastavat elementi algandmetes ei eksisteeri. Töö käigus nullitakse sagedustabelis järjest elemente, mis on täielikult analüüsitud. Sellisel moel keelatud/elimineeritud elemente ei lisata edaspidi leitavatesse lõigetesse; lõigete moodustamisel lähevad arvesse vaid nullist (või kõrgemast sageduslävest) suurema sagedusega elemendid.

MONSA tulemuseks on lõigete hulk (mis vastab suletud hulkadele[1]) ja/või puud (mets). Tulemused esitatakse leidmise järjekorras, see järjestus ei sõltu objektide esialgsest järjestusest. Tippude (lõigete) sagedused vähenevad rangelt piki puu harusid (juurest lehtede suunas). See vähenemine võimaldab harusid kärpida vastavalt ette antud sageduslävele (minimaalne lubatud sagedus). Asjaolu, et sageduste vähenemine on range, annab lõikele (tippude/elementide kombinatsioonile alates juurest kuni jooksva tipuni) suure potentsiaali olla suletud hulk. Igal tasemel leitakse ühise vanema järglastipud nende sageduste nõrgalt kahanevas järjestuses. Puude juured leitakse samuti sageduste nõrgalt kahanevas järjestuses. Võrdse sagedusega tippude leidmise/valimise järjekord sõltub otsingu printsiibist (tavaliselt piki sagedustabeli veerge või piki selle ridu).

Olemuselt on MONSA rekursiivne algoritm. Siinkohal esitame selle tagurdamist kasutava versiooni (vastavalt artiklile (Kuusik & Lind 2008)).

Pseudokoodis kasutame järgmisi tähistusi:

t – rekursiooni tase (sügavus)

FT_t – hulga X_t sagedustabel

$Lõige_t$ – lõige üle hulga X_t (elementide vektor)

Algoritm MONSA

1: Algväärtustamine
2: $t \leftarrow 0$, $Lõige_0 \leftarrow \{\}$
3: Leida FT_0
4: DO WHILE leidub $FT_s \# Ø$ IN $\{FT_s\}$, $s \leq t$
5: FOR EACH element $h_f \in FT_t$ sagedusega $V=\max FT_t(h_f) \# 0$ DO
6: IF on vaja kärpida (h_f) THEN GOTO Tagurda
7: Eraldada alamtabel $X_{t+1} \subset X_t$ nii et $X_{t+1}=\{Xij \in X_t \mid X.f=h_f\}$
8: Leida FT_{t+1}
9: NullidAlla(t+1)
10: KontrolliUnikaalsust(t+1)
11: IF uus lõige on unikaalne THEN
12: Lisa elemendid j sagedusega V ($FT_{t+1}(j)=V$) vektorisse $Lõige_{t+1}$
13: Tagasivõrdlus(t+1)
14: Väljasta $Lõige_{t+1}$

[1] Suletud hulk (*closed set*) - on objektide hulga ühiste elementide maksimaalne hulk. Mistahes elemendi lisamine suletud hulgale vähendab selle katet ja sagedust.

15: IF leidub analüüsiks sobivaid tunnuseid THEN $t \leftarrow t+1$
16: ENDIF
17: NEXT
18: Tagurda: $t \leftarrow t-1$
19: $Lõige_{t+1} \leftarrow Lõige_t$
20: ENDDO
21: Kõik lõiked on leitud
Algoritmi lõpp

PROCEDURE NullidAlla(t+1)
FOR EACH element $h_u \in FT_t$ DO
IF $FT_t(h_u)=0$ THEN $FT_{t+1}(h_u) \leftarrow 0$
NEXT
END PROCEDURE

PROCEDURE Tagasivõrdlus(t+1)
FOR EACH element $h_u \in FT_{t+1}$ sagedusega #0 DO
IF $FT_{t+1}(h_u)=FT_t(h_u)$ THEN $FT_t(h_u) \leftarrow 0$
NEXT
END PROCEDURE

PROCEDURE KontrolliUnikaalsust(t+1)
IF hulgal X_{t+1} leidub element h_u, $1 \le u \le M$ nii et
[$h_u \in Lõige_{t+1}$ AND $FT_{t+1}(h_u)=0$ AND h_u sagedus hulgal $X_{t+1}=V$] THEN
lõige ei ole unikaalne
ELSE
lõige on unikaalne
ENDIF
END PROCEDURE

MONSA on sügavuti otsingu algoritm, mis tagurdab siis, kui jooksev haru on ammendatud või seda tuleb kärpida. Algoritmi põhisammud igal (rekursiooni)tasemel:

S1: Valida juhtelement – esimene element maksimaalse sagedusega (üle nulli; vähemalt kasutaja poolt määratud lävesagedusega) (rida 5), lisada see element (potentsiaalsesse) lõikesse and nullida vastav lahter sagedustabelis

S2: Leida järgmine sagedustabel objektidele, mis sisaldavad juhtelementi (rida 8)

S3: Kui leidub element(e), mille sagedus on võrdne juhtsagedusega, kontrollida lõike unikaalsust (rida 10) => kui see on unikaalne, lisada need elemendid (millede sagedus on võrdne juhtsagedusega) lõikesse (rida 12); vastasel korral tagurdada

S4: Väljastada lõige (rida 14)

S5: Eelmise taseme sagedustabeli nullid “alla tuua” st jooksval tasemel nullida nende elementide sagedused, millede sagedus eelmisel tasemel on nullitud (rida 9)

S6: “Tagasivõrdlus”: eelmisel tasemel nullida nende elementide sagedus, millede sagedused eelmisel ja jooksval tasemel on võrdsed (rida 13)

Korduste (s.o juba leitud lõigete permutatsioonide) leidmise ära hoidmiseks kasutatakse MONSAs järgmisi elimineerimistehnikaid:

1) “nullide alla toomine” – tegevus, mis takistab juba leitud lõike väljastamise järgmis(t)el (sügavama(te)l) taseme(te)l;
2) “tagasivõrdlus” – tegevus, mis keelab leitud lõike väljastamise samal (jooksval) tasemel ja ka kõrgema(te)l taseme(te)l (pärast tagurdamist);
3) “unikaalsuskontroll” – tõkestab juba leitud lõigete alamlõigete väljastamise.

Kahe esimese tehnika mõju on tõestatud artiklis (Kuusik 1993) (vastavalt teoreemid 5.3 ja 5.4). Unikaalsuskontrolli selgitatakse lähemalt artiklis (Kuusik & Lind 2008). Ilma nende

elimineerimistehnikateta leiaks algoritm kõigi eksisteerivate väärtuskombinatsioonide kõik permutatsioonid.

Näide

Illustreerimaks algoritmi tööpõhimõtteid paremini, peame valima teistsuguse lähtetabeli, kui korrastusmeetodite korral. Aga et korrastusmeetodite tulemused oleksid võrreldavad hüpoteeside generaatori (HG) tulemustega, väljastame käesoleva näite järel ka HG tulemused korrastusmeetodite näitetabelil.

Olgu antud algtabel:

X_0	*A1*	*A2*	*A3*
1.	1	0	3
2.	2	2	1
3.	2	3	0
4.	2	0	2
5.	0	1	3
6.	0	1	3
7.	1	1	2
8.	1	0	3
9.	2	3	0

Seame leitavate lõigete sageduspiiriks SP=2, st väljastame kõik lõiked, mille esinemissagedus on vähemalt 2. Rakendame eelpool kirjeldatud algoritmi MONSA.

Leiame algtabeli tunnuste väärtuste esinemissagedused sagedustabelisse FT_0 (FT_t, t=0). (rida 3)

FT_0	A1	A2	A3
0	2	3	2
1	3	3	1
2	4	1	2
3	0	2	4

S1: Valime suurima sagedusega elemendi juhttipuks. Kui neid on mitu, valime positsiooniliselt esimese. Antud näites on selliseid kaks: A1.2 ja A3.3. Positsiooniliselt on esimeseks tunnuse A1 väärtus 2 sagedusega 4 (A1.2=4; V=4). Nullime selle esinemissageduse sagedustabelis (FT_0(A1.2)=4→0).

FT_0	A1	A2	A3
0	2	3	2
1	3	3	1
2	0	1	2
3	0	2	4

Jooksev tase on t+1=0+1=1. Lisame A1.2 lõikesse: $Lõige_1$=A1.2. Teeme väljavõtu X_1, st eraldame uude tabelisse kõik objektid, mis sisaldavad elementi A1.2 (tegelikult me ei pea sellist tabelit füüsiliselt tekitama, piisab, kui jätame meelde vastavad tabeli rea numbrid kui indeksid):

X_1: *A1.2*	*A1*	*A2*	*A3*
2.	2	2	1
3.	2	3	0
4.	2	0	2
9.	2	3	0

S2: Leiame tabelile X_1 vastavad esinemissagedused FT_1, seejuures juhttipule vastav tunnus (A1) lülitatakse edasistes tegevustes (t+1, t+2 jne) välja:

FT_1	A1	A2	A3
0		1	2
1		0	1
2		1	1
3		2	0

S3: Kontrollime, kas sagedustabelis leidub elemendi sagedust=V=4. Sellist ei ole, seega me ei saa lõiget mingi(te) uu(t)e elementidega laiendada. S4: Väljastame lõike **L1: A1.2=4**.

S5: Järgmisena kanname eelmise sagedustabeli FT_0 sagedused=0 positsiooniliselt jooksvasse sagedustabelisse FT_1 (hetkel selliseid uusi „0" pole). S6: Seejärel teeme tagasivõrdluse. Selleks võrdleme hetkel vaatluse all olevat sagedustabelit (t+1=1) positsiooniliselt eelmise taseme (t=0) sagedustabeliga. Kui mingi positsiooni sagedused on võrdsed (värvitud kollaseks), siis eelmise taseme sagedustabelis vastav sagedus nullitakse.

FT_0	A1	A2	A3
0	2	3	2
1	3	3	1
2	0	1	2
3	0	2	4

FT_1	A1	A2	A3
		1	2
		0	1
		1	1
		2	0

Uus

FT_0	A1	A2	A3
0	2	3	0
1	3	3	0
2	0	0	2
3	0	0	4

t=t+1=0+1=1. S1: Valime sagedustabelist FT_1 juhttipu. Meil on kaks kandidaati A2.3=2 ja A3.2=2, mõlema sagedus rahuldab kehtestatud sageduspiiri SP≥2. Valime juhttipuks positsiooniliselt esimese, A2.3, lisame selle lõikesse: $Lõige_2$: A1.2 & A2.3. V=2. Nullime juhttipule vastava sageduse:

FT_1	A1	A2	A3
0		1	2
1		0	1
2		1	1
3		0	0

Teeme väljavõtu A2.3 järgi:

X_2: A2.3	*A1*	*A2*	*A3*
3.	2	3	0
9.	2	3	0

S2: Leiame esinemissagedused, seejuures juhttipule vastav tunnus lülitatakse edasistes tegevustes (t+1, t+2 jne) välja:

FT_2	A1	A2	A3
0			2
1			0
2			0
3			0

S3: Kontrollime, kas sagedustabelis FT_2 leidub elementi, mille sagedus=V=2. Selline leidub, A3.0=2. See tähendab, et saame lõiget laiendada, kuid peame kontrollima, kas formeeruv lõige on unikaalne. Selleks võrdleme A3.0 sagedust tasemel 2 (sagedustabelis FT_2) selle sagedusega tasemel 1 (FT_1). Kui need on võrdsed, siis on tegu originaalse lõikega, kui mitte, siis korduva lõikega. $FT_2(A3.0)=FT_1(A3.0)=2$, seega on formeeruv lõige originaalne. Lisame A3.0 lõikesse: $Lõige_2$: A1.2&A2.3&A3.0=2, väljastame lõike **L2: A1.2&A2.3&A3.0=2** (S4).

S5: Kanname FT_1 sagedused=0 sagedustabelisse FT_2, kusjuures FT_2 sisu ei muutu.
S6: Teeme tagasivõrdluse:

FT_1	A1	A2	A3
0		1	2
1		0	1
2		1	1
3		0	0

FT_2	A1	A2	A3
		0	2
		0	0
		0	0
		0	0

Uus

FT_1	A1	A2	A3
0		1	0
1		0	1
2		1	1
3		0	0

Nullime A3.0 esinemissageduse FT_2-s:

FT_2	A1	A2	A3
0			0
1			0
2			0
3			0

Kuna sagedustabel FT_2 on tühi, siis ühtegi uut juhttippu valida pole. Jätkame tasemel 1, kus $Lõige_1$=A1.2.

FT_1	A1	A2	A3
0		1	0
1		0	1
2		1	1
3		0	0

S1: Kontrollime, kas leidub tippe, millede sagedus ≥SP. Selliseid pole, seega juhttippu valida ei saa. Tagurdame: t=t-1=1-1=0 (rida 18).

FT_0	A1	A2	A3
0	2	3	0
1	3	3	0
2	0	0	2
3	0	0	4

S1: Valime juhttipu, suurim sagedus A3.3=4 >SP, kanname A3.3 lõikesse: $Lõige_1$=A3.3. V=4. Nullime juhttipu esinemissageduse FT_0-s:

FT_0	A1	A2	A3
0	2	3	0
1	3	3	0
2	0	0	2
3	0	0	0

Teeme väljavõtu:

X_1: *A3.3*	*A1*	*A2*	*A3*
1.	1	0	3
5.	0	1	3
6.	0	1	3
8.	1	0	3

S2: Leiame sagedused:

FT_1	A1	A2	A3
0	2	2	
1	2	2	
2	0	0	
3	0	0	

S3: Kontrollime, kas sagedustabelis FT_1 leidub elementi sagedusega 4, kuid ei leidu – seega lõiget laiendada ei saa – väljastame lõike **L3: A3.3=4** (S4).

S5: Kanname eelmise sagedustabeli FT_0 nullid FT_1-sse, FT_1 ei muutu. S6: Teeme tagasivõrdluse:

FT_0	A1	A2	A3
0	2	3	0
1	3	3	0
2	0	0	2
3	0	0	0

FT_1	A1	A2	A3
	2	2	
	2	2	
	0	0	
	0	0	

Uus

FT_0	A1	A2	A3
0	0	3	0
1	3	3	0
2	0	0	2
3	0	0	0

t=t+1=0+1=1. $Lõige_1$=A3.3, V=4.

FT_1-s on meil neli juhttipu kandidaati sagedusega=2 ≥SP. Valime positsiooniliselt esimese: A1.0=2, V=2. Lisame selle lõikesse: $Lõige_2$=A3.3 & A1.0=2. Nullime selle esinemissageduse FT_1-s:

FT_1	A1	A2	A3
0	0	2	
1	2	2	
2	0	0	
3	0	0	

Teeme väljavõtu:

X_2: *A1.0*	*A1*	*A2*	*A3*
5.	0	1	3
6.	0	1	3

Formeerime sagedustabeli (S2):

FT_2	A1	A2	A3
0		0	
1		2	
2		0	
3		0	

S3: Kontrollime, kas tabelis leidub sagedust=V=2. Leidub: A2.1=2. Kontrollime formeeruva lõike originaalsust: FT_1(A2.1)=FT_2(A2.1)=2, seega lõige originaalne, lisame elemendi lõikesse: $Lõige_2$=A3.3 & A1.0 & A2.1=2, väljastame lõike **L4: A3.3 & A1.0 & A2.1=2** (S4).

Kanname FT_1 nullid alla, FT_2 ei muutu (S5). Teeme tagasivõrdluse (S6):

FT_1	A1	A2	A3
0	0	2	
1	2	2	
2	0	0	
3	0	0	

FT_2	A1	A2	A3
		0	
		2	
		0	
		0	

Uus

FT_1	A1	A2	A3
0	0	2	
1	2	0	
2	0	0	
3	0	0	

Nullime A2.0 sageduse: $FT_2(A2.0)=0$:

FT_2	A1	A2	A3
0		0	
1		0	
2		0	
3		0	

Sagedustabel on tühi. Jätkame tasemel 1, kus $Lõige_1$=A3.3.

FT_1	A1	A2	A3
0	0	2	
1	2	0	
2	0	0	
3	0	0	

S1: Valime juhttipu: kaks kandidaati, mõlema sagedus≥SP. Valime esimese: A1.1=2. V=2. Lisame selle lõikesse: $Lõige_2$=A3.3 & A1.1. Nullime selle esinemissageduse $FT_1(A1.1)=0$.

FT_1	A1	A2	A3
0	0	2	
1	0	0	
2	0	0	
3	0	0	

Teeme väljavõtu:

X_2:			
A1.1	*A1*	*A2*	*A3*
1.	1	0	3
8.	1	0	3

Leiame sagedused (S2):

FT_2	A1	A2	A3
0		2	
1		0	
2		0	
3		0	

S3: Kontrollime, kas tabelis leidub sagedust=V=2. Leidub: A2.0=2. Kontrollime formeeruva lõike originaalsust: $FT_1(A2.0)=FT_2(A2.0)=2$, seega lõige on originaalne, lisame elemendi lõikesse: $Lõige_2$=A3.3 & A1.1 & A2.0=2, väljastame lõike **L5: A3.3 & A1.1 & A2.0=2** (S4).
Kanname FT_1 nullid FT_2-sse (S5). FT_2 ei muutu. Teeme tagasivõrdluse (S6):

FT_1	A1	A2	A3
0	0	2	
1	0	0	
2	0	0	
3	0	0	

FT_2	A1	A2	A3
		2	
		0	
		0	
		0	

Uus

FT_1	A1	A2	A3
0	0	0	
1	0	0	
2	0	0	
3	0	0	

Nullime A2.0 sageduse: $FT_2(A2.0)=0$:

FT_2	A1	A2	A3
0		0	
1		0	
2		0	
3		0	

Sagedustabel FT_2 on tühi, jätkame tasemel 1.

FT_1	A1	A2	A3
0	0	0	
1	0	0	
2	0	0	
3	0	0	

Sagedustabel FT_1 on tühi, tagurdame: t:=t-1=1-1=0.

FT_0	A1	A2	A3
0	0	3	0
1	3	3	0
2	0	0	2
3	0	0	0

S1: Valime juhttipu: kolm kandidaati, kõigi sagedus=3 ≥SP. Valime esimese: A1.1=3. Lisame selle lõikesse: $Lõige_1$=A1.1. V=3. Nullime selle esinemissageduse FT_0(A1.1)=0.

FT_0	A1	A2	A3
0	0	3	0
1	0	3	0
2	0	0	2
3	0	0	0

Teeme väljavõtu A1.1 järgi:

X_1:			
A1.1	*A1*	*A2*	*A3*
1.	1	0	3
7.	1	1	2
8.	1	0	3

S2: Leiame sagedused:

FT_1	A1	A2	A3
0		2	0
1		1	0
2		0	1
3		0	2

S3: Kontrollime, kas tabelis FT_1 leidub sagedust=V=3. Ei leidu, väljastame lõike **L6: A1.1=3** (S4).

S5: Kanname eelmise taseme nullid sagedustabelist FT_0 vaadeldavasse sagedustabelisse FT_1:

FT_0	A1	A2	A3
0	0	3	0
1	0	3	0
2	0	0	2
3	0	0	0

FT_1	A1	A2	A3
		2	0
		1	0
		0	1
		0	2

Uus

FT_1	A1	A2	A3
0	0	2	0
1	0	1	0
2	0	0	1
3	0	0	0

S6: Teeme tagasivõrdluse, kokkulangevusi pole:

FT_0	A1	A2	A3
0	0	3	0
1	0	3	0
2	0	0	2
3	0	0	0

FT_1	A1	A2	A3
		2	0
		1	0
		0	1
		0	0

Uus

FT_0	A1	A2	A3
0	0	3	0
1	0	3	0
2	0	0	2
3	0	0	0

t=t+1=0+1=1. Lõige$_1$=A1.1.

S1: Leiame FT$_1$-st juhttipu, selleks on A2.0=2 ≥SP. Lisame selle lõikesse: Lõige$_2$=A1.1 & A2.0=2. V=2.

FT$_1$	A1	A2	A3
0		2	0
1		1	0
2		0	1
3		0	0

Nullime A2.0 esinemissageduse: FT$_1$(A2.0)=2→0:

FT$_1$	A1	A2	A3
0		0	0
1		1	0
2		0	1
3		0	0

Teeme väljavõtu A2.0 järgi:

X$_2$: *A2.0*	*A1*	*A2*	*A3*
1.	1	0	3
8.	1	0	3

Leiame sagedused (S2):

FT$_2$	A1	A2	A3
0			0
1			0
2			0
3			2

S3: Kontrollime, kas tabelis leidub sagedust=V=2. Leidub – A3.3. Kontrollime formeeruva lõike originaalsust: FT$_2$(A3.3)=2 ≠ FT$_1$(A3.3)=0, seega saadav lõige pole originaalne (muidu saavutaksime lõike L5: A3.3 & A1.1 & A2.0=2, mis on korra juba eraldatud).

Jätkame tasemel 1, Lõige$_1$= A1.1.

FT$_1$	A1	A2	A3
0		0	0
1		1	0
2		0	1
3		0	0

Kuna kõikide analüüsis olevate elementide sagedused on <SP, siis tagurdame: t:=t-1=1-1=0.

FT$_0$	A1	A2	A3
0	0	3	0
1	0	3	0
2	0	0	2
3	0	0	0

S1: Valime juhttipu: kaks kandidaati, kõigi sagedus=3 ≥SP. Valime esimese: A2.0=3. Lisame selle lõikesse: Lõige$_1$=A2.0. V=3.

Nullime selle esinemissageduse FT_0(A2.0)=0.

FT_0	A1	A2	A3
0	0	0	0
1	0	3	0
2	0	0	2
3	0	0	0

Teeme väljavõtu A2.0 järgi:

X_1: *A2.0*	*A1*	*A2*	*A3*
1.	1	0	3
4.	2	0	2
8.	1	0	3

Leiame sagedused (S2):

FT_1	A1	A2	A3
0	0		0
1	2		0
2	1		1
3	0		2

S3: Kontrollime, kas uues tabelis FT_1 leidub sagedust=V=3. Ei leidu, lõiget laiendada ei saa, väljastame lõike: **L7: A2.0=3** (S4).

Kanname eelmise taseme sagedustabeli FT_0 nullid sagedustabelisse FT_1 (S5):

FT_0	A1	A2	A3
0	0	0	0
1	0	3	0
2	0	0	2
3	0	0	0

FT_1	A1	A2	A3
	0		0
	2		0
	1		1
	0		2

Uus

FT_1	A1	A2	A3
0	0		0
1	0		0
2	0		1
3	0		0

Teeme tagasivõrdluse, see ei rakendu (S6). Kuna alles jäänud elementide sagedus FT_1-s <SP, siis juhttippu valida ei saa. Jätkame tasemel 0.

FT_0	A1	A2	A3
0	0	0	0
1	0	3	0
2	0	0	2
3	0	0	0

(S1) Valime juhttipu: selleks on A2.1=3, lisame selle lõikesse: Lõige$_1$=A2.1 =3. V=3. Nullime vastava sageduse sagedustabelis FT_0:

FT_0	A1	A2	A3
0	0	0	0
1	0	0	0
2	0	0	2
3	0	0	0

Teeme väljavõtu:

X_1:			
A2.1	*A1*	*A2*	*A3*
5.	0	1	3
6.	0	1	3
7.	1	1	2

Leiame sagedused FT_1 (S2):

FT_1	A1	A2	A3
0	2		0
1	1		0
2	0		1
3	0		2

Kontrollime, kas uues tabelis FT_1 leidub sagedust=V=3. Ei leidu, lõiget laiendada ei saa, väljastame lõike: **L8: A2.1=3** (S4).

Kanname eelmise taseme sagedustabeli FT_0 nullid sagedustabelisse FT_1 (S5):

FT_0	A1	A2	A3
0	0	0	0
1	0	0	0
2	0	0	2
3	0	0	0

FT_1	A1	A2	A3
	2		0
	1		0
	0		1
	0		2

Uus

FT_1	A1	A2	A3
0	0		0
1	0		0
2	0		1
3	0		0

Teeme tagasivõrdluse, see ei rakendu (S6). Kuna alles jäänud elementide sagedus FT_1-s <SP, siis juhttippu valida ei saa. Jätkame tasemel 0.

FT_0	A1	A2	A3
0	0	0	0
1	0	0	0
2	0	0	2
3	0	0	0

(S1) Leiame FT_0-st juhttipu, ainuke kandidaat A3.2=2 ≥SP. Kanname selle lõikesse: $Lõige_1$=A3.2. V=2. Nullime vastava sageduse FT_0-s:

FT_0	A1	A2	A3
0	0	0	0
1	0	0	0
2	0	0	0
3	0	0	0

Teeme väljavõtu X_1:

X_1:			
A3.2	*A1*	*A2*	*A3*
4.	2	0	2
7.	1	1	2

Leiame sagedused FT_1 (S2):

FT_1	A1	A2	A3
0	0	1	
1	1	1	
2	1	0	
3	0	0	

(S3) Kontrollime, kas uues tabelis FT_1 leidub sagedust=V=2. Ei leidu, lõiget laiendada ei saa, väljastame lõike: **L9: A3.2=2** (S4). Kanname eelmise taseme sagedustabeli FT_0 nullid sagedustabelisse FT_1 (S5):

FT_0	A1	A2	A3
0	0	0	0
1	0	0	0
2	0	0	0
3	0	0	0

FT_1	A1	A2	A3
	0	1	
	1	1	
	1	0	
	0	0	

Uus

FT_1	A1	A2	A3
0	0	0	
1	0	0	
2	0	0	
3	0	0	

Teeme tagasivõrdluse, see ei rakendu (S6). Kuna sagedustabel on tühi, juhttippu valida ei saa. Jätkame tasemel 0.

FT_0	A1	A2	A3
0	0	0	0
1	0	0	0
2	0	0	0
3	0	0	0

Näeme, et sagedustabel FT_0 on tühi. Et rohkem tagurdada ei saa, lõpetame töö, kõik lõiked on leitud:

L1: A1.2=4
L2: A1.2 & A2.3 & A3.0=2
L3: A3.3=4
L4: A3.3 & A1.0 & A2.1=2
L5: A3.3 & A1.1 & A2.0=2
L6: A1.1=3
L7: A2.0=3
L8: A2.1=3
L9: A3.2=2

Et saada mingitki võrdlusaspekti korrastusmeetoditega, väljastame hüpoteeside generaatori (HG) rakendamise tulemuse andmetabeli korral, millel selgitasime korrastusmeetodeid:

i/j	*1*	*2*	*3*	*4*	*5*
1.	1	0	0	0	0
2.	0	1	0	1	1
3.	0	1	0	1	1
4.	1	1	0	1	0
5.	0	0	1	0	1
6.	0	1	1	1	1

HG tulemused korrastusmeetodite näitetabelil. Minimaalne lubatud sagedus = 1.

Leitud lõiked:

```
T1.0&T5.1=4
T1.0&T5.1&T2.1&T4.1=3
T1.0&T5.1&T2.1&T4.1&T3.0=2
T1.0&T5.1&T2.1&T4.1&T3.1=1
T1.0&T5.1&T3.1=2
T1.0&T5.1&T3.1&T2.0&T4.0=1
T2.1&T4.1=4
T2.1&T4.1&T3.0=3
T2.1&T4.1&T3.0&T1.1&T5.0=1
T3.0=4
T3.0&T1.1&T5.0=2
T3.0&T1.1&T5.0&T2.0&T4.0=1
T2.0&T4.0=2
```

Näeme, et iga lõige kirjeldab mingit ühtemoodi käituvat elementide rühma andmetabelis. Teame, et korrastusmeetodite korral muutus korrastatud andmetabel oluliselt informatiivsemaks, kuna nähtavaks muutus andmestu sisemine struktuur. Samas, kui andmetabel on piisavalt suur, ei suuda inimsilm kõike haarata ja palju olulist jääb märkamata. HG väljastatud lõiked toovad aga esile kõik ette antud sageduspiirile vastavad mustrid ehk objekt-tunnus süsteemi allsüsteemid ehk ühtemoodi käituvad elementide alamhulgad. Siin tekib aga jällegi probleem: mida selle infohulgaga teha? Asi muutub oluliselt paremini hoomatavamaks, kui väljastame HG tulemuse puu kujul:

```
Uus puu
(4)       0.750(3)    0.667(2)
T1.0&T5.1=>T2.1&T4.1->T3.0
                      0.333(1)
                      ->T3.1
          0.500(2)0.500(1)
          =>T3.1   ->T2.0&T4.0

Uus puu
(4)       0.750(3)0.333(1)
T2.1&T4.1=>T3.0   ->T1.1&T5.0

Uus puu
(4) 0.500(2)    0.500(1)
T3.0=>T1.1&T5.0->T2.0&T4.0

Uus puu
(2)
T2.0&T4.0
```

Kuidas interpreteerida HG-puud? Esimene tipp selles on alati nn juurtipp („Uus puu”), järgnevad tipud samal tasandil tuleb interpreteerida kui „ja” seost, näiteks: T1.0 ja T5.1 ja T2.1 ja T4.1 ja T3.0. Tipu kohal sulgudes on vastava lõike esinemissagedus, teiseks suuruseks on, kui suure osa moodustab järgmise tipu lisamisel saadud lõike objektide arv eelmise lõike objektide arvust, st see on kahe järjestikuse lõike esinemissageduste suhe. Näiteks: 3/4=0,750; 2/3=0,667 jne. Lisaks „ja” suhtele kajastub HG-puus ka „või” suhe, näiteks: (T1.0 ja T5.1 ja T2.1 ja T4.1 ja) T3.0 või (T1.0 ja T5.1 ja T2.1 ja T4.1 ja) T3.1. Teine näide: (T1.0 ja T5.1 ja) T2.1 ja T4.1 ja T3.0 või (T1.0 ja T5.1 ja) T3.1 ja T2.0 ja T4.0.

Algoritmi samm-sammulist tööd demonstreerib video: hüpoteeside generaator.

Determinatsioonanalüüs

Järgnevalt tutvustame determinatsioonanalüüsi (DA) meetodit, mille töötas välja vene teadlane Dr Sergei Tšesnokov. Oleme seda meetodit edasi arendanud, luues uusi võimalusi, mida originaalmeetodis ei leidunud. Oma olemuselt on tegu masinõppe valdkonna meetodiga, mis võimaldab teatud omadustega objektide hulga (Y) tarbeks leitud reegleid kasutada selle objektide hulga kirjeldamiseks. Kirjeldamise põhiküsimuseks on: Kes nad on? Mis on neile omane? Mis eristab neid teistest?

Järgnevates peatükkides tutvustame meetodi olemust ja meetodi erinevate omadustega versioone.

8 Determinatsioonanalüüsi põhimõisted

DA põhimõisted esitame vastavalt Tšesnokovi töödele (*Чесноков* 1980, 1982, 2002).

Kui omadusega X kaasneb alati omadus Y, siis eksisteerib reegel X→Y (kui X siis Y). Sellist seost X ja Y vahel nimetatakse determinatsiooniks (X-st Y-sse). X on determineeriv (determineerija) ja Y determineeritav.

Determinatsioonil X→Y on viis karakteristikut:

- n(X) – nende objektide arv, millel on omadus X
- n(XY) – nende objektide arv, millel on nii omadus X kui ka omadus Y
- n(Y) – nende objektide arv, millel on omadus Y
- A(X→Y) = n(XY) / n(X) – determinatsiooni täpsus (*accuracy*)
- C(X→Y) = n(XY) / n(Y) – determinatsiooni täielikkus (*completeness*)

Determinatsiooni täpsus näitab, mil määrab X determineerib Y-i, st kui suur osa omadusega X objektidest kuulub omadusega Y objektide hulka.

Determinatsiooni täielikkus näitab, kui suur osa Y-st on determineeritud X-i poolt, st kui palju Y objektidest sisaldavad omadust X.

Mõlema näitaja väärtused jäävad vahemikku 0..1 (0%..100%). Väärtuse 1 korral on determinatsioon **täiesti täpne**, st kõik objektid omadusega X kuuluvad ainult omadusega Y kirjeldatud objektide hulka, või **täielik**, st kõik objektid omadusega Y sisaldavad omadust X.

Omadus X koosneb faktoritest. Faktoriks on konkreetne tunnus konkreetse väärtusega. Iga tunnuse kohta on nii mitu erinevat faktorit, kui on sel tunnusel erinevaid väärtusi.

Lisades reeglisse $X \rightarrow Y$ uue faktori Z, saame uue reegli $XZ \rightarrow Y$, mille täpsus ja täielikkus võivad erineda esialgse omast. Uue faktori panust reegli täpsusesse/täielikkusesse mõõdetakse reegli täpsuse/täielikkuse juurdekasvuga:

- $\Delta A(Z) = A(XZ \rightarrow Y) - A(X \rightarrow Y)$ – faktori Z panus reegli $XZ \rightarrow Y$ täpsusesse
- $\Delta C(Z) = C(XZ \rightarrow Y) - C(X \rightarrow Y)$ – faktori Z panus reegli $XZ \rightarrow Y$ täielikkusesse

Panus täpsusesse võib olla vahemikus -1..1. Vastavalt sellele jaotatakse faktorid positiivseteks e olulisteks ($\Delta A(Z)>0$ – Z lisamine muudab reegli täpsemaks), negatiivseteks ($\Delta A(Z)<0$ – Z lisamisel reegli täpsus väheneb) ja ebaolulisteks e nullfaktoriteks ($\Delta A(Z)=0$ – Z lisamine ei mõjuta reegli täpsust).

Täpne reegel ei sisalda negatiivseid faktoreid. Kui reegel koosneb ainult positiivsetest faktoritest, nimetatakse seda normaalseks reegliks.

Faktori panus reegli täielikkusesse võib olla negatiivne ($\Delta C(Z)<0$, st Z lisamine vähendab reegli arvukust) või null ($\Delta C(Z)=0$, st Z lisamine ei muuda reegli arvukust).

Reeglite süsteem on reeglite hulk $S_q = \{X_i \rightarrow Y \mid i=1,2,...,q\}$, kus q on reeglite arv. Reeglisüsteemi iseloomustavad keskmine täpsus, summaarne täielikkus ja summaarne võimsus (reeglitega kaetud objektide koguarv).

Reeglisüsteem S_q on aditiivne, kui reeglid X_i paarikaupa ei lõiku/ülekattu (st ei kata samu objekte). Aditiivse süsteemi täielikkus ja võimsus saadakse reeglite täielikkuste C_i ja võimsuste $n(X_i)$ summeerimisel. Reeglite täpsusi ei saa liita.

Reeglisüsteem on täpne, kui kõik selle reeglid on täpsed ($A(X_i \rightarrow Y) = 1$).

Determinatsioonanalüüsi peamiseks ülesandeks on leida omadusest X omadusse Y kõik determinatsioonid, millel on vähemalt minimaalne lubatud täpsus ja minimaalne lubatud täielikkus. Ideaaljuhul maksimaalselt täpne ja maksimaalselt täielik reeglisüsteem (*Чесноков* 1982). Sellise süsteemi saab leida ainult juhul, kui andmetes pole vastuolusid st olukorda, kus ühesuguse eeldusosaga (X) objektid on erineva järeldusega (Y).

9 Kuidas kasutada determinatsioonanalüüsi

DA kasutusmetoodikat kirjeldame järgmisel näitel.

Näide

Oletame, et oleme määratlenud omadused X ja Y.

Omadus X:
Tunnus T1: suhtlemisvajaduse realiseerimine
1 – piisavalt
2 – mittepiisavalt
3 – raske öelda/ei tea

Omadus Y:
Tunnus T2: tööalased eelistused
1 – eelistab huvitavat tööd kõrgele palgale
2 – eelistab tasuvamat, kuid vähem huvitavat tööd
3 – raske öelda/ei tea

Esitame vastavad andmed sagedustabelina, olgu selleks järgmine tabel:

X (T1) \ Y (T2)	*1*	*2*	*3*	*Kokku*
1	32 41%	23 29%	24 30%	79 100%
2	9 35%	6 23%	11 42%	26 100%
3	11 23%	14 30%	22 47%	47 100%
Kokku	52 34%	43 28%	57 38%	152 100%

Uurime valimit Y=1, s.o „kes on need inimesed, kes eelistavad huvitavat tööd kõrgele palgale".
|Y=T2.1| = 52
Vastus: huvitavat tööd eelistavad kõrgele palgale need, kes

1) on piisavalt realiseerinud suhtlemisvajadust
   $A(X\rightarrow Y) = 32/79=0,41$; $C(X\rightarrow Y) = 32/52 = 0,62$.
2) ei ole piisavalt realiseerinud suhtlemisvajadust
   $A(X\rightarrow Y) = 9/26=0,35$; $C(X\rightarrow Y) = 9/52 = 0,17$.
3) raske öelda/ei tea
   $A(X\rightarrow Y) = 11/47=0,23$; $C(X\rightarrow Y) = 11/52 = 0,21$.

Kuna ühegi determinatsiooni täpsus ei võrdu 1, siis peame lisaks omadusele X analüüsi juurde tooma mingi teise tunnuse Z (millise, otsustab kasutaja). Tunnuste lisamine toimub senikaua, kuni determinatsiooni täielikkus=100% (st Y saab 100% kaetud reeglitega, milledel täpsus=1).

10 Determinatsioonanalüüsi originaalmeetodi käsitlus

Determinatsioonanalüüsi originaalmeetodi korral leitakse kõik sellised reeglid $X_i\rightarrow Y$, milles X_i sisaldab kõiki etteantud tunnuseid. Seega on kõik reeglid ühepikkused. Selline reeglisüsteem on aditiivne. Reeglid ei lõiku omavahel ja iga objekti jaoks saab olla maksimaalselt üks reegel.

Igale sisendile vastab täpselt üks reeglisüsteem. Teistsuguse reeglisüsteemi saamiseks tuleb sisendit muuta.

Sisendiks on X tunnuste loeteluna ja Y – ühe tunnuse üks konkreetne väärtus. Lisaks saab ette anda piiranguid reegli ja faktori tasemel.

Väljund esitatakse tabelina, mille iga rida esitab üht reeglit (determinatsiooni). Iga reegli $X_i\rightarrow Y$ kohta esitatakse kõigepealt faktorid, millest X koosneb, ja nende järel determinatsiooni karakteristikud: $A(X\rightarrow Y)$, $C(X\rightarrow Y)$, n(X), n(XY), n(Y). Neist viimane on ühesugune kõigi sama Y-t määravate determinatsioonide jaoks, seega võib seda näidata ühe korra (mitte kõigil ridadel).

(Iga reegli) iga faktori kohta leitakse selle panus täpsusse ΔA ja panus täielikkusse ΔC. Panused arvutatakse kõigi teiste faktorite suhtes (sõltumata nende esitamise järjekorrast).

Lisaks leitakse terve reeglisüsteemi kohta selle (keskmine) täpsus, täielikkus (reeglite täielikkuste summa), n(X) summa ja n(XY) summa.

Kasutaja saab seada piiranguid reeglite täpsusele ja täielikkusele ning faktorite panusele täpsusesse ja panusele täielikkusesse. Kui vähemalt üks piirangutest ei kehti, jääb reegel tulemusest välja.

Kui võimalikud piirangud ei kõrvalda ühtki reeglit, on leitud süsteem täielik – iga objekti jaoks leidub (täpselt üks) reegel. See süsteem sisaldab tunnuste X kõiki eksisteerivaid väärtuskombinatsioone, iga kombinatsioon on ühe reegli eeldusosaks. Piirangute kasutamisel ei pruugi leitud reeglisüsteem olla täielik.

Et reeglisüsteem oleks täpne (koosneks ainult täpsetest reeglitest), tuleb seada reegli täpsusele piirang 100%. Selline süsteem võib olla mittetäielik (C < 100%).

Et leida reegleid, mis koosnevad ainult positiivsetest (ja ebaolulistest) faktoritest, saab seada faktori panusele täpsusesse piirangu >0 (≥0).

Selline lähenemine ei võimalda leida reeglisüsteeme, milles reeglid oleksid erineva pikkusega ja võiksid lõikuda.

Näide

Olgu meil järgmised andmed (Quinlan, 1984):

i/j	*Height*	*Hair*	*Eyes*	*Class*
1.	*tall*	*dark*	*blue*	*–*
2.	*short*	*dark*	*blue*	*–*
3.	*tall*	*blond*	*blue*	*+*
4.	*tall*	*red*	*blue*	*+*
5.	*tall*	*blond*	*brown*	*–*
6.	*short*	*blond*	*blue*	*+*
7.	*short*	*blond*	*brown*	*–*
8.	*tall*	*dark*	*brown*	*–*

Kui eesmärgiks on determineerida *Class*.– tunnuste *Eyes* ja *Hair* abil, saame täpse ja täieliku (aditiivse) süsteemi:

- *Eyes.blue & Hair.dark* → *Class*.– (C = 40%)
- *Eyes.brown & Hair.dark* → *Class*.– (C = 20%)
- *Eyes.brown & Hair.blond* → *Class*.– (C = 40%)

Originaaltarkvara DA-System (lühidalt DAS) 4.0 esitab tulemuse tabelina:

Y: Class.– **N(Y)=5**						
	Selgitavad tunnused		**Reeglisüsteemi karakteristikud**			
	Eyes	***Hair***	**A**	**C**	**N(X)**	**N(XY)**
Reegel 1	*blue*	*dark*	1,00	0,40	2	2
ΔA	0,00	0,60				
ΔC	-0,20	0,00				
Reegel 2	*brown*	*dark*	1,00	0,20	1	1
ΔA	0,00	0,00				
ΔC	-0,40	-0,40				
Reegel 3	*brown*	*blond*	1,00	0,40	2	2
ΔA	0,50	0,00				
ΔC	0,00	-0,20				
Reeglisüsteemi summaarsed karakteristikud:			1,00	1,00	5	5
Läved:	0 < A <= 1	-1 <= ΔA <= 1	0 <= C <= 1		-1 <= ΔC <= 1	

Selline originaalväljund annab meile infot faktorite panuste kohta. Tunnuste esitamise järjekord ei oma sisulist tähtsust. Panused on arvutatud (sama reegli) kõigi ülejäänud faktorite suhtes. Kui reegli vasakul poolel oleks 4 faktorit, siis oleks nt 1. faktori panus arvutatud nii, nagu oleks 2., 3. ja 4. faktor olemas ja siis lisataks see esimene. Ja kõigi teistega samamoodi.

Panus täpsusse ΔA näitab, kui palju suurendab konkreetne faktor reegli täpsust. Võtame näiteks reegli 3, mille täpsus on 1 (A=1). ΔA(*Eyes.brown*)=0,50 näitab, et ilma faktorita *Eyes.brown* oleks reegli täpsus 0,50 võrra väiksem s.o A(*Hair.blond*→*Class*.–)=1-0,5=0,5. ΔA(*Hair.blond*)=0 näitab, et reegel oleks täpne ka ilma selle faktorita: A(*Eyes.brown*→*Class*.–)=1-0=1. Seega on *Hair.blond* antud reeglis ebaoluline (klassi tuvastamise seisukohalt). Reeglis 2 on mõlemad faktorid nullfaktorid, see näitab, et (klassi tuvastamiseks) piisab emmast-kummast.

Panus täielikkusse ΔC näitab, kui palju suurendab konkreetne faktor reegli täielikkust. See panus ei saa kunagi positiivne olla, sest iga uue faktori lisamisel kaetavate objektide arv n(XY) väheneb või jääb samaks. Reegli 2 puhul on mõlema faktori panuseks -0,40, mis näitab, et kahe faktoriga reegel katab 40% vähem objekte kui (emma-kumma) ühe faktoriga. Nt ΔC(*Eyes.brown*)=-0,4 näitab, et C(*Hair.dark*→*Class*.–)=0,2-(-0,4)=0,6. Reeglis 3 ΔC(*Eyes.brown*)=0, mis tähendab, et selle faktori lisamisel ei muutu (antud klassi kuuluvate) kaetud objektide arv: C(*Hair.blond*→*Class*.–)=0,4-0=0,4.

DAS on põhjalikumalt kirjeldatud artiklis (Lind & Kuusik 2007), abiks olid DALSolutioni (2007) ja Contexti materjalid (*Контекст* 1998, 1999).

11 Samm-sammuline lähenemine

Võrreldes originaalse lähenemisega, võimaldab samm-sammuline lähenemine leida erineva pikkusega reegleid, säilitades aditiivsuse (reeglite mittelõikumise). Kui reeglid on lühemad, jääb neist välja osa ebaolulisi faktoreid (nullfaktoreid), see vähendab liiasust. Endiselt on kitsenduseks maksimaalselt üks reegel objekti kohta.

Oluliseks muutub tunnuste järjekord. Tunnuseid lisatakse kõigisse reeglitesse samas järjekorras. Kui reegel osutub täpseks (A=1), siis seda enam ei laiendata. Teistesse reeglitesse lisatakse tunnuseid edasi, kuni ka need saavad täpseks (või lõpevad tunnused otsa). Saadud väljund vastab otsustuspuule.

Erinevad tunnuste järjekorrad annavad üldjuhul erinevad tulemused. Võimalike erinevate järjestuste arv on faktoriaal tunnuste arvust. Tunnuste järjekorra saab ette anda või määrata sõltuvalt töö käigus leitavast infost kas automaatselt või kasutaja poolt (interaktiivse liidese korral).

Panuseid saab arvutada vaid jooksva reegliosa suhtes, mitte (veel-mitte-teadaoleva) lõpliku reegli suhtes.

Algoritm

Pseudokoodis kasutame järgmisi tähistusi:

- tunnused – reeglites kasutatavate tunnuste hulk
- pot – potentsiaalsete reeglite hulk
- uus_pot – järgmise iteratsiooni potentsiaalsete reeglite hulk
- vastus – leitud reeglite hulk

Määratle Y, tunnused

```
vastus ← ∅, C_kokku ← 0
pot ← {[]}
Leia n(Y)
FOR EACH tunnus IN tunnused DO
        uus_pot ← ∅
        FOR EACH reegel IN pot DO
                FOR EACH väärtus OF tunnus DO
                        X = reegel & tunnus.väärtus
                        leia n(X), n(X Y), A(X→Y), C(X→Y)
                        IF A(X→Y)=1 THEN
                                vastus ← vastus ∪ {X→Y}
                                C_kokku ← C_kokku + C(X→Y)
                                IF C_kokku=1 THEN GOTO Lõpp
                        ELSEIF A(X→Y)>0 AND A(X→Y)<1 THEN
                                uus_pot ← uus_pot ∪ X
                        ELSE
                                //XY ei eksisteeri
                        ENDIF
                NEXT väärtus
        NEXT reegel
        pot ← uus_pot
NEXT tunnus
Lõpp. Kõik reeglid on leitud
```

Algoritm on avaldatud artiklis (Lind & Kuusik 2008a).

Näide

Näites kasutame sama tabelit (Quinlan 1984).

i/j	*Height*	*Hair*	*Eyes*	*Class*
1.	*tall*	*dark*	*blue*	–
2.	*short*	*dark*	*blue*	–
3.	*tall*	*blond*	*blue*	+
4.	*tall*	*red*	*blue*	+
5.	*tall*	*blond*	*brown*	–
6.	*short*	*blond*	*blue*	+
7.	*short*	*blond*	*brown*	–
8.	*tall*	*dark*	*brown*	–

Eesmärgiks on determineerida *Class*.+ (kirjeldada isikuid, kes kuuluvad sellesse klassi). Selles klassis on 3 objekti (isikut): n(Y)=3. Olgu tunnuste lisamise järjekorraks nende esitamise järjekord algtabelis: 1) *Height*, 2) *Hair*, 3) *Eyes*.

Esiteks leitakse reeglid, mis koosnevad vaid atribuudist *Height*:

Height	n(X)	n(XY)	A	C	ΣC
short	3	1	1/3	1/3	
tall	5	2	2/5	2/3	

Kumbki kahest (kandidaat)reeglist pole täpne. Seega tuleb lisada mõlemasse järgmine tunnus – *Hair*. Teoreetiliselt on 6 erinevat kombinatsiooni tunnuste *Height* ja *Hair* väärtustest. Tegelikkuses üht neist (*Height.short*&*Hair.red*) ei eksisteeri (n(X)=0) ja kaks kombinatsiooni (*Height.short*&*Hair.dark*; *Height.tall*&*Hair.dark*) on sellised, mis ei esine antud klassis (n(XY)=0). Need kolm jäetakse analüüsist välja (kuna need reaalses andmestus ei eksisteeri, siis neid ei leita, st töö lähtub reaalsest andmestust, teoreetilisvõimalikke kombinatsioone ei leita).

Height	*Hair*	n(X)	n(XY)	A	C	ΣC
short	*dark*	1	0	0	0	
short	*red*	0				
short	*blond*	2	1	1/2	1/3	
tall	*dark*	2	0	0	0	
tall	***red***	**1**	**1**	**1**	**1/3**	1/3
tall	*blond*	2	1	1/2	1/3	

Kolm järelejäänud reeglit esinevad klassis Y. Üks neist (*Height.tall*&*Hair.red*) on täpne (A=1) ega vaja rohkem faktoreid. See reegel katab 1/3 klassist (C=1/3). Täpne reegel läheb tulemusse. Täpsete reeglite täielikkused C summeeritakse (ΣC) eesmärgiga jõuda 100%-lise katteni.

Kaht reeglit, mille täpsus jääb 0 ja 1 vahele, laiendatakse järgmise tunnusega (*Eyes*):

Height	Hair	Eyes	n(X)	n(XY)	A	C	ΣC
short	***blond***	***blue***	**1**	**1**	**1**	**1/3**	2/3
short	*blond*	*brown*	1	0	0	0	
tall	***blond***	***blue***	**1**	**1**	**1**	**1/3**	1
tall	*blond*	*brown*	1	0	0	0	

Neljast reeglist kaks on täpsed (A=1), mõlema täielikkus on 1/3. Oleme leidnud 3 täpset reeglit klassile „+"kogutäielikkusega 100% (ΣC=1). Samal ajal on kahe ülejäänud reegli täpsus 0 ja neid me ei laienda. Seega on klass täielikult kirjeldatud (reeglisüsteem S1):

- *Height.tall&Hair.red* → *Class*.+ (C=1/3)

- *Height.short&Hair.blond&Eyes.blue* → *Class*.+ (C=1/3)
- *Height.tall&Hair.blond&Eyes.blue* → *Class*.+ (C=1/3)

Kasutades teistsugust tunnuste järjekorda, võime saada teistsuguse tulemuse. Näiteks järjestuse 1) *Hair*, 2) *Eyes*, 3) *Height* korral saame 2 täpset reeglit, mis katavad klassi „+" täielikult (reeglisüsteem S2):

- *Hair.red* → *Class*.+ (C=1/3)
- *Hair.blond&Eyes.blue* → *Class*.+ (C=2/3)

Nagu näha, tunnus *Height* pole siin klasside eristamiseks vajalik.

Analüüsi seisukohalt („kes nad on?") tähendab see, et uuritavat valimit Y (antud juhul *Class*.+) saab kirjeldada mitut moodi: kas kirjeldus S1 või kirjeldus S2. Kumba eelistada, jääb kasutaja enda otsustada. Võimalus valida Y kirjeldamiseks mingid reeglid mõlemast reeglihulgast pole korrektne, sest nõudeks on, et valitud reeglitega peab Y olema kaetud 100% (ΣC=1).

Algoritmi samm-sammulist tööd demonstreerib video: determinatsioonanalüüs.

12 Esimene lõikuvate reeglite algoritm

Mittelõikuvate reeglitega (s.o aditiivne) süsteem põhimõtteliselt ei võimalda vabaneda liiasusest nullfaktorite näol (kuigi mõne väikese näite puhul võib see nii olla). Seepärast on mõistlik lubada reeglitel lõikuda, samuti kasutada erinevates reeglites erinevat tunnuste järjekorda. Reeglite pikkused (faktorite arv reeglis) võivad olla erinevad.

Esimene lõikuvate reeglite algoritm leiab võimalikult väikese reeglite hulga, jälgides ja arvestades objektide kaetust. Potentsiaalne reegel lisatakse tulemusse vaid juhul, kui see katab vähemalt üht veel katmata objekti.

Tulemuseks on üks mitte-aditiivne (s.o lõikuvate reeglitega) süsteem. Kui algandmetes pole vastuolusid, on leitud süsteem täpne (koosneb vaid täpsetest reeglitest) ja täielik (kõik objektid on kaetud). Vastuolu on olukord, kus samasuguse kirjeldusega objektid kuuluvad erinevatesse klassidesse, kasutatavatest tunnustest ei piisa nende eristamiseks.

Sõltuvalt faktorite valiku põhimõttest võime saada samade andmetega erinevad reeglisüsteemid.

Võrreldes aditiivsete reeglisüsteemidega, on reeglite arv tavaliselt väiksem ja reeglid lühemad. Siiski ei garanteeri see lähenemine lühimate võimalike reeglite leidmist, samuti valimi Y objektide kaetust vähima reeglite arvuga.

Algoritm

Määratleda X and Y

Samm 0. t:=0; U_t:=∅

IF kõik Y-sse kuuluvad objektid on kaetud THEN GOTO Lõpp

Samm 1. Leida sagedused tabelites X_t ja Y_t: Fx_t, Fy_t

Samm 2. FOR EACH faktor A, mille korral $Fy_t(A)=Fx_t(A)$ ja A katab mõnda veel katmata objekti

väljasta reegel $\{U_i\}$&A, i=0,…,t

IF leiti vähemalt üks uus reegel THEN GOTO Samm0

Samm 3. Valida vaba faktor U_t

IF selliseid faktoreid ei leidu THEN

$\{U_i\}$, i=0,…,t on vastuolu;

GOTO Samm0

t:=t+1; teha väljavõtt objektidest, mis sisaldavad faktorit U_t;

GOTO Samm1

Lõpp. Reeglisüsteem on leitud

Et leida reegli täpsust, vajame kaht sagedust: n(XY) ja n(X). Igal iteratsioonil **t** (objektidest väljavõtu tegemisel) kogume need kaks sagedust kõigi faktorite kohta kahte sagedustabelisse: $\mathbf{Fx_t}$ (sagedused n(X)) ja $\mathbf{Fy_t}$ (sagedused n(XY)) (Samm1). $\mathbf{Fx_t}$ ja $\mathbf{Fy_t}$ kokku moodustavad nn 3D-sagedustabeli. Nende abil saame iga faktori kohta leida selle reegli täpsuse, mille saame antud faktori lisamisel reeglisse.

Täiendavalt kasutatakse veel sagedustabelit $\mathbf{Fc_t}$, milles peetakse arvet reeglitega (juba) kaetud objektide üle. Sagedusi tabelis **Fc** uuendatakse iga kord, kui leitakse uus reegel. **Fc** abil saame kindlaks teha 1) millised faktorid on vabad, 2) kas potentsiaalne reegel katab mõnd veel katmata objekti ning 3)

kas võib töö lõpetada. Alternatiivselt võib kaetud objektide sageduste **Fc** asemel kasutada (Y-sse kuuluvate) vabade (s.o veel katmata) objektide sagedusi (**Fy**-**Fc**).

Allpool seletame sageduste kasutamist lähemalt.

Tegemist on rekursiivse algoritmiga. Igal tasemel **t** valitakse faktor $\mathbf{U_t}$, tehakse väljavõtt $\mathbf{X_t}$ objektidest, mis sisaldavad faktorit $\mathbf{U_t}$ (Samm3), ja leitakse sellele vastavad sagedustabelid $\mathbf{Fx_t}$, $\mathbf{Fy_t}$, $\mathbf{Fc_t}$ (Samm1).

Igas väljavõtus (mistahes tasemel) kehtib: kui mingi faktor esineb vaid ühe klassi objektidel, siis saame selle faktori lisamisel (juba varem selekteeritud faktoritele, mis on ühised kõigile jooksva väljavõtu $\mathbf{X_t}$ objektidele) täpse reegli. Sellise faktori sagedused on võrdsed sagedustabelites $\mathbf{Fx_t}$ ja $\mathbf{Fy_t}$ (võrdsete sageduste korral on täpsus 1). Kui selline uus reegel katab vähemalt ühte (klassi Y kuuluvat) objekti, mis pole veel reeglitega kaetud, lisatakse see reegel tulemusse. Seda tingimust saab kontrollida, kasutades kaetud objektide sagedusi ($\mathbf{Fc_t}$). Kui faktori sagedus **Fc**-s on sama nagu **Fx**-s ja **Fy**-s, siis on kõik seda faktorit sisaldavad objektid juba reeglitega kaetud ja selle faktoriga reeglit tulemusse ei lisata, kui aga sagedus **Fc**-s on väiksem, on uus reegel sobiv. Sel viisil leitakse iga reegli viimane faktor (Samm2).

Kõik faktorid enne viimast valitakse rekursiivselt jooksvast (alam)hulgast $\mathbf{X_t}$ (Samm3). Faktori vabadust saame kontrollida **Fc** abil: faktori sagedus tabelis $\mathbf{Fc_t}$ peab olema väiksem kui $\mathbf{Fy_t}$-s. Valik (vabade faktorite hulgast) tehakse sageduste baasil.

Valitakse maksimaalse vaba sagedusega ($\mathbf{Fy_t}$-$\mathbf{Fc_t}$) faktor, võrdsete maksimaalsete väärtuste korral eelistatakse faktorit, millel on suurem sagedus ka **Fx**-s.

Iga kord peale uu(t)e reegli(te) leidmist pöördub algoritm tagasi algtasemele ja alustab jälle esimese faktori valimisega. Võrreldes (võimaliku) tagasipöördumisega eelmisele tasemele, saame leida võimalikult lühikesed (väheste faktoritega) reeglid.

Algoritm võimaldab kindlaks teha ka vastuolusid – olukordi, kus erinevatesse klassidesse kuuluvad objektid on ühesuguse kirjeldusega (kasutatud atribuutide ulatuses). See olukord saabub siis, kui pole enam ühtki faktorit, mille järgi saaks teha järgmise väljavõtu st kõik tunnused on juba kasutatud, kuid täpsus on <1 (Samm3). Ka sellised vastuolulised objektid loetakse kaetuks tabeli **Fc** uuendamisel.

Fc abil saame ka kindlaks teha, millal lõpetada töö. Töö võib lõpetada siis, kui kõik Y-sse kuuluvad objektid on reeglitega kaetud. Sellisel juhul on sagedused tabelis $\mathbf{Fc_0}$ võrdsed sagedustega tabelis $\mathbf{Fy_0}$. Muidugi piisab lihtsalt sellest, et kõik Y-sse kuuluvad objektid on kaetud (või vastuolulised).

Kui algtabelis esineb vastuolusid, pole võimalik saavutada täielikku katet (s.o maksimaalset täielikkust) täpsete reeglitega. Üks variant on väljastada ka vastuolusid kajastavad mitte-täpsed reeglid (näidates nende madalama täpsuse). Sel juhul pole reeglisüsteem enam täpne.

Algoritm on avaldatud artiklis (Kuusik & Lind 2010).

Näide

Algoritmi demonstreerimiseks kasutame jälle Quinlani andmeid (1984), seekord kodeeritud kujul. Järgnev tabel esitab kasutatud kodeeringut.

Tunnus	*Height*	*Hair*	*Eyes*	*Class*
Kood	***1***	***2***	***3***	***4***
1	*short*	*dark*	*blue*	–
2	*tall*	*red*	*brown*	+
3		*blond*		

Kodeeritud algandmed:

i \ j	*1*	*2*	*3*	*4*
1.	2	1	1	1
2.	1	1	1	1
3.	2	3	1	2
4.	2	2	1	2
5.	2	3	2	1
6.	1	3	1	2
7.	1	3	2	1
8.	2	1	2	1

Olgu X X(8,3), X_{ij} = 1,...,3 ja Y=4.1 {Y_i: 1,2,5,7,8} (s.o *Class*.–). Leiame esialgsed sagedustabelid Fx ja Fy (Samm1):

Fx	K_j \ *j*	*1*	*2*	*3*	*Fy*	K_j \ *j*	*1*	*2*	*3*
	1	3	3	5		1	2	3	2
	2	5	1	3		2	3	0	3
	3	0	4	0		3	0	2	0

Faktori 2.1 (tunnus 2 väärtusega 1) sagedused Fx-s ja Fy-s on võrdsed. See tähendab, et kõik objektid, mis sisaldavad faktorit 2.1, kuuluvad klassi Y. Saame reegli 2.1=3 (*Hair.dark*→*Class*.–). Pärast „=" märki on reegli sagedus, mis näitab, et antud reegel katab 3 objekti (nimelt objektid 1, 2 ja 8). Ka faktori 3.2 sagedused Fx-s ja Fy-s on võrdsed. Saame reegli 3.2=3 (*Eyes.brown*→*Class*.–), mis katab samuti 3 objekti (objektid 5, 7 ja 8), sh selliseid, mis pole eelmise reegliga kaetud. (Samm2)

Nende kahe reegliga

- *Hair.dark*→*Class*.– (3 objekti)
- *Eyes.brown*→*Class*.– (3 objekti)

on kõik klassi Y objektid kaetud (Samm0). Seejuures objekti 8 katavad mõlemad reeglid, siin ilmneb reeglite kattumine/lõikumine.

Demonstreerimaks algoritmi teisi samme, leiame nüüd teise klassi (*Class*.+) reeglid. Seekord Y=4.2 {Y_i: 3,4,6}. Esialgsed sagedustabelid Fx and Fy (Samm1):

Fx_0	K_j \ *j*	*1*	*2*	*3*	Fy_0	K_j \ *j*	*1*	*2*	*3*
	1	3	3	5		1	1	0	3
	2	5	1	3		2	2	1	0
	3	0	4	0		3	0	2	0

Samm 2. Faktori 2.2 sagedused tabelites Fx and Fy on võrdsed. Saame reegli 2.2=1 (*Hair.red*→*Class*.+), mis katab objekti 4.

Seejärel suurendatakse (esialgseid nulliseid) sagedusi kaetud objektide sagedustabelis Fc:

Fc_0	K_j \ *j*	*1*	*2*	*3*
	1	0	0	1
	2	1	1	0
	3	0	0	0

või alternatiivselt leitakse vabad sagedused Fy-s (reeglitega katmata Y-sse kuuluvate objektide sagedused):

$Ffree_0$	K_j \ *j*	*1*	*2*	*3*
	1	1	0	2
	2	1	0	0
	3	0	2	0

Pöördume tagasi sammu 0. Kõik objektid ei ole veel kaetud (seda näitab ka: Fc_0 ei lange kokku Fy_0-ga või $Ffree_0$ sisaldab nullist suuremaid sagedusi). Kuna sellel tasemel rohkem reegleid pole, jõuame sammu 3.

Uue reegli alustamiseks leiame maksimaalse vaba sagedusega faktori (Ffree järgi). Et neid on kaks, vaatame samade faktorite sagedusi Fx-s. Faktori 3.1 sagedus on 5 ja faktori 2.3 sagedus 4. Eelistades suurema Fx sagedusega faktorit, valime faktori 3.1. Teeme objektidest väljavõtu 3.1 järgi:

i \ j	*1*	*2*	*3*	*4*
1.	2	1	1	1
2.	1	1	1	1
3.	2	3	1	2
4.	2	2	1	2
6.	1	3	1	2

Selle väljavõtu sagedustabelid (Samm1):

Fx_1	K_j \ *j*	*1*	*2*	*3*
	1	2	2	5
	2	3	1	0
	3	0	2	0

Fy_1	K_j \ *j*	*1*	*2*	*3*
	1	1	0	3
	2	2	1	0
	3	0	2	0

Fc_1	K_j \ *j*	*1*	*2*	*3*
	1	0	0	1
	2	1	1	0
	3	0	0	0

Samm2. Fx ja Fy põhjal näeme kaht potentsiaalset reeglit: (3.1&) 2.2 ning (3.1&) 2.3. Sagedustabelis Fc on faktori 2.2 sagedus sama mis Fx-s ja Fy-s. See tähendab, et objektid, mis sisaldavad faktorit 2.2, on juba reeglitega kaetud. Seetõttu sellist reeglit tulemusse ei lisata (reegel 3.1&2.2=1 olekski liiane, kattes sama objekti kui reegel 2.2=1). Faktori 2.3 sagedus Fc-s on väiksem kui 2 (Fx-s ja Fy-s), seega sobib see faktor reegli moodustamiseks. Saame teise reegli: 3.1&2.3=2 (*Eyes.blue*&*Hair.blond*→*Class*.+), see katab objekte 3 ja 6.

Pöördudes tagasi algtasemele (Samm0), värskendame tabelit Fc:

Fc_0	K_j \ *j*	*1*	*2*	*3*
	1	1	0	3
	2	2	1	0
	3	0	2	0

Võrreldes tabelit Fc esialgse Fy-ga, näeme, et kõik sagedused neis on võrdsed – järelikult võime töö lõpetada. Tõepoolest, kaks leitud reeglit

- *Hair.red* → *Class*.+ (1 objekt)
- *Eyes.blue&Hair.blond*→*Class*.+ (2 objekti)

katavad kõik klassi Y (4.2) kuuluvad objektid. Antud klassi kuuluvad objektid on võimalik katta mittelõikuvate mitteliiaste reeglitega.

13 Determineeriv Reeglite Hulk

Selle asemel, et leida üks reeglisüsteem, võiks leida kõik liiasusi mitte sisaldavad reeglid ja nendest moodustada vastavalt vajadusele erinevaid katteid (reeglihulki). Selleks on vaja algoritmi, mis leiaks (vähemalt) kõik niisugused reeglid, ja protseduuri, mis kõrvaldaks liigsed reeglid (kui algoritmi tulemus neid sisaldab). Vastav algoritm on toodud järgmises alapeatükis.

Siinkohal tutvustame nn determineerivat reeglite hulka.

Võrdleme kaht täpset reeglit:

- *Eyes.blue & Hair.dark* → *Class*.– (C = 40%; 2 objekti)
- *Hair.dark* → *Class*.– (C = 60%; 3 objekti)

Olles huvitatud võimalikult lühikestest reeglitest, eelistame teist reeglit. See katab neid kaht objekti, mida ka esimene reegel, ja lisaks veel üht objekti. Niisugusel juhul ütleme, et esimene reegel sisaldub teises ehk on teise reegli alamreegel. Võrreldes reeglite vasakuid pooli näeme, et esimene on pikem, sisaldades kõiki teise reegli faktoreid ja mõningaid täiendavaid faktoreid. Niisugusel juhul loeme esimese reegli liigseks.

Olgu antud tabel X(N,M) ja (ainult) klassi Y kirjeldavate kõikvõimalike reeglite hulk B, kus iga reegel esineb vaid üks kord.

Klassi Y determineeriv reeglite hulk (DRH) koosneb kõigist sellistest hulga B reeglitest, mis ei sisaldu hulga B teistes reeglites.

B = $\{R_i\}$, i=1, 2,..., K, kus K on kõikvõimalike (ainult) klassi Y kirjeldavate reeglite hulk.

$R_i \neq R_j$, $i \neq j$.

DRH = $\{R_u\}$. $R_u \in$ DRH kui /$\exists$ $R_i \in$ B, $R_u \subset R_i$, $i \neq u$. DRH $\subseteq$ B

See tähendab, et DRH ei sisalda oma reeglite alamreegleid. B-st DRH saamiseks peame kõik alamreeglid välja viskama. Seda protsessi nimetame reeglite kompresseerimiseks.

Näide. Koosnegu B 4 reeglist (kõikvõimalikud reeglid klassile Y=Class.1):

- r1: IF T1.1 & T2.1 THEN Class.1
- r2: IF T1.1 & T3.2 THEN Class.1
- r3: IF T2.1 THEN Class.1
- r4: IF T3.2 THEN Class.1

Nagu näha, r1 sisaldub r3-s ja r2 r4-s. Vastavalt definitsioonile DRH_B = {r3, r4}.

DRH peamised omadused on:

1. reeglid ei sisalda liiaseid faktoreid/tunnuseid (nullfaktoreid),
2. klassi Y kuuluv objekt võib olla kaetud mitme reegliga.

Sellist kompresseerimist saab kasutada nii eraldi protseduurina pärast põhialgoritmi (mis leiab potentsiaalseid DRH reegleid) kui ka põhialgoritmi töö ajal, iga kord, kui leitakse uus reegel. Kompresseerimise hõlbustamiseks tuleks silmas pidada:

- Liigne reegel on pikem (selle vasak pool sisaldab rohkem faktoreid) kui see reegel, mis antud reegli välja tõrjub;
- Selle liigse reegli vasak pool sisaldab kõiki neid faktoreid mis lühem reegel;
- Selle reegli sagedus $\leq$ lühema reegli sagedus;
- Mõlemad reeglid kirjeldavad sama klassi;
- Pikem (liiane) reegel leitakse alati enne kui see lühem.

Viimasena nimetatud omadus on iseloomulik nii MS algoritmidele kui ka paljudele teistele algoritmidele.

Kompresserimisprotseduuri saab rakendada sõltumata sellest, kas reeglihulk sisaldab kõiki vajalikke reegleid või mitte. Kui ei sisalda, siis ei saa ka lõpptulemus neid sisaldada.

Meie algoritm kõigi DRH jaoks vajalike reeglite leidmiseks (mida tutvustame järgmises peatükis) leiab võimalikult vähe liigseid reegleid.

DRH baasil saame püstitada ja lahendada järgmisi ülesandeid – leida:

1. Lühimad reeglid (faktorite arvu poolest),
2. Pikimad reeglid (faktorite arvu poolest),
3. Kindlate omadustega reeglid, nt kõik kindla pikkusega reeglid DRH-s,
4. Lühim reeglisüsteem (s.o vähima arvu reeglitega süsteem),
5. Süsteem, mis koosneb minimaalse pikkusega reeglitest,
6. Kõikvõimalikud reeglisüsteemid, mida saab DRH baasil moodustada.

Ülesanded 1–3 on kergesti lahendatavad. Ülesanded 4–5 on NP-täielikud. Kõik need ülesanded on olulised reeglite järeltöötluse jaoks.

DRH leidmine on töömahukas ja ei pruugi sobida kiireks ühekordseks info kogumiseks, kuid annab sobiva baasi reeglite järeltöötlemise jaoks.

DRH on avaldatud artiklis (Kuusik & Lind 2011).

14 Kõigi võimalikult lühikeste reeglite leidmise algoritm

Algoritm leiab kõik need reeglid, mida on vaja DRH jaoks s.o kõik reeglid, mis ei sisaldu teistes reeglites ja mõned liiased reeglid, mis kõrvaldatakse kompresseerimisega. Reeglid võivad lõikuda/ ülekattuda, sõltumata sellest, kui mitmekordselt objektid juba kaetud on. Algoritm väldib nii palju kui võimalik liiasust (nullfaktorite näol), leides võimalikult lühikesed reeglid.

Erinevalt eelmisena esitatud 3D-algoritmist (esimene lõikuvate reeglite algoritm), kasutatakse siin tavapärast tagurdamist eelmisele tasemele (mitte algtasemele), ei jälgita objektide kaetust ning kasutatakse MONSAst tuttavat „nullide alla toomise" tehnikat.

Algoritm

Määratleda X and Y

Samm0. t:=0; U_t:=∅

Samm1. Leida sagedused tabelites X_t ja Y_t: Fx_t, Fy_t
IF t>0 THEN
FOR EACH faktor A, mille korral $Fy_{t-1}(A)=0$
$Fy_t(A) \leftarrow 0$

Samm2. FOR EACH faktor A, mille korral $Fy_t(A)=Fx_t(A)$
väljasta reegel $\{U_i\}$&A, i=0,…,t
$Fy_t(A) \leftarrow 0$

Samm3. IF pole piisavalt vabu faktoreid väljavõtu tegemiseks THEN
IF t=0 THEN GOTO Lõpp
t:=t-1; GOTO Samm3

Samm4. Valida kasutamata faktor U_t
$Fy_t(A) \leftarrow 0$
t:=t+1; teha väljavõtt objektidest, mis sisaldavad faktorit U_t
GOTO Samm1

Lõpp. Reeglisüsteem on leitud

Tegemist on sügavuti otsinguga, algoritm teeb järjestikuseid väljavõtte valitud faktorite järgi. Igal tasemel tehakse kõigepealt kindlaks reeglid, mis sellelt tasemelt tulevad (Samm2), ja seejärel ükshaaval faktorid, millede järgi tehakse väljavõtud (Samm4).

Kasutatakse kaht sagedustabelit: $\mathbf{X_t}$ (väljavõtu kõik objektid) jaoks $\mathbf{Fx_t}$ ja $\mathbf{Y_t}$ (need väljavõtu objektid, mis kuuluvad Y-sse) jaoks $\mathbf{Fy_t}$. Kui mõne faktori sagedus on võrdne mõlemas tabelis, siis see faktor lõpetab reegli, mis sisaldab ka kõiki eelmiste väljavõttude aluseks olevaid faktoreid.

Väljavõtu tegemiseks kasutatavad faktorid valitakse sageduste järgi, esmalt maksimaalse sageduse järgi **Fy**-s, ja kui neid faktoreid on rohkem kui üks, siis väiksema sageduse järgi **Fx**-s. Kui (jooksvas väljavõtus) on „vabu" (kasutamata) väärtusi (mida näitab nullist suurem sagedus **Fy**-s) vaid ühel tunnusel, siis pole mõtet teha järgmist väljavõttu, sest selles väljavõtus poleks vabu faktoreid, mille abil eristada eri klassidesse kuuluvaid objekte. Kui vabu faktoreid üldse pole (st pole nullist suuremaid sagedusi), siis mõistagi pole võimalik väljavõttu teha. Mõlemal juhul tagurdab algoritm eelmisele tasemele (Samm3).

Iga faktori, mida on kasutatud kas reegli lõpetamiseks (Samm2) või väljavõtu tegemiseks (Samm4), sagedus nullitakse jooksva taseme **Fy**-s. Iga **Fy** (v.a algtasemel) pärib kõik eelmise taseme nullid (kutsume seda „nullide alla toomiseks") (Samm1). Need nullimistehnikad hoiavad ära palju liigseid väljavõtte ja reegleid, kaotamata samas DRH jaoks vajalikke reegleid.

Erinevused võrreldes esimese lõikuvate reeglite algoritmiga:

- Objektide kaetust (Fc abil) ei jälgita;
- Pärast reegli leidmist ei tagurdata algtasemele
- Kasutatakse elimineerimistehnikat „nullide alla toomine"
- Vastuolusid ei tuvastata

Algoritm on avaldatud artiklis (Kuusik & Lind 2011). Algoritmi laiendades on võimalik leida reeglid kõigi klasside jaoks (Kuusik & Lind 2012).

Näide

Näites kasutame jälle Quinlani andmeid (1984) kodeeritud kujul:

i \ j	*1*	*2*	*3*	*4*
1.	2	1	1	1
2.	1	1	1	1
3.	**2**	**3**	**1**	**2**
4.	**2**	**2**	**1**	**2**
5.	2	3	2	1
6.	**1**	**3**	**1**	**2**
7.	1	3	2	1
8.	2	1	2	1

Koodide tähendused on esitatud leheküljel 66.

Olgu X X(8,3), X_{ij} = 1,...,3 ja Y=4.2 {Y_i: 3,4,6} (s.o *Class*.+). Leiame esialgsed sagedustabelid Fx ja Fy (Samm1):

Fx_0	K_j \ *j*	*1*	*2*	*3*	Fy_0	K_j \ *j*	*1*	*2*	*3*
	1	3	3	5		1	1	0	3
	2	5	1	3		2	2	1	0
	3	0	4	0		3	0	2	0

Samm2: Faktori 2.2 (*Hair.red*) sagedused tabelites Fx ja Fy on võrdsed, mis tähendab, et kõik objektid, milles on faktor 2.2, kuuluvad klassi Y. Siit saame reegli R1: 2.2=1 (*Hair.red*). Sagedus (pärast võrdusmärki) näitab, et reegel katab ühe objekti (nimelt objekti 4). Faktori 2.2 sagedus Fy-s nullitakse, et edaspidi selle faktori järgi väljavõttu mitte teha. Sagedustabelite seis on nüüd selline:

Fx_0	K_j \ *j*	*1*	*2*	*3*	Fy_0	K_j \ *j*	*1*	*2*	*3*
	1	3	3	5		1	1	0	**3**
	2	5	*0*	3		2	2	*0*	0
	3	0	4	0		3	0	2	0

Samm4: Järgmiseks tuleb valida faktor, mille alusel teha väljavõtt. Valime faktori 3.1 (*Eyes.blue*), millel on suurim sagedus Fy-s. Väljavõtt (s.o X-i alamtabel) 3.1 järgi:

i \ j	*1*	*2*	*3*	*4*
1.	2	1	1	1
2.	1	1	1	1
3.	**2**	**3**	**1**	**2**
4.	**2**	**2**	**1**	**2**
6.	**1**	**3**	**1**	**2**

ja sellele vastavad sagedused (Samm1):

Fx_1	K_j \ *j*	*1*	*2*	*3*	Fy_1	K_j \ *j*	*1*	*2*	*3*
	1	2	2	5		1	1	0	3
	2	3	*0*	0		2	2	*0*	0
	3	0	2	0		3	0	2	0

Samm1: Iga Y-le vastav sagedustabel (Fy) pärib kõik eelmise taseme nullid („nullide alla toomine“), et vältida korduvaid väljavõtte ja liiaseid reegleid. Seepärast asendatakse faktori 2.2 tegelik sagedus (=1) nulliga. Kui me seda ei teeks, saaksime reegli 3.1&2.2=1, mis on juba leitud reegli 2.2=1 alamreegliks. Samm2: Sellest väljavõtust saame reegli R2: 3.1&2.3=2 (*Eyes.blue* & *Hair.blond*). Nullime faktori 2.3 sagedused.

Samm3: Nüüd on olukord selline, et kõik nullist suuremad sagedused Fy-s – 1.1 ja 1.2 – asuvad samas veerus. Ükskõik kumma järgi teeksime väljavõtu, ei saa me sellest reeglit leida, sest pole rohkem tunnuseid, mille järgi (järgmisel tasemel) klasse eristada. Seetõttu tagurdame eelmisele tasemele (antud juhul on selleks algtase). Sagedustabelid algtasemel:

Fx_0	$K_j \setminus j$	*1*	*2*	*3*	Fy_0	$K_j \setminus j$	*1*	*2*	*3*
	1	3	3	*0*		1	1	0	*0*
	2	5	*0*	3		2	**2**	*0*	0
	3	0	**4**	0		3	0	**2**	0

Selle taseme sagedustabelis Fy on juba 2 sagedust nullitud: 2.2 siis, kui leidsime selle järgi reegli, ja 3.1 seetõttu, et tegime selle alusel väljavõtu. Nüüd on Fy-s kaks maksimaalse sagedusega (=2) faktorit: 1.2 ja 2.3. Et nende sagedused Fx-s on erinevad, valime faktori 2.3, mille sagedus Fx-s on väiksem. Väljavõtt 2.3 järgi ja sellele vastavad sagedused:

i \ j	*1*	*2*	*3*	*4*
3.	**2**	**3**	**1**	**2**
5.	2	3	2	1
6.	**1**	**3**	**1**	**2**
7.	1	3	2	1

Fx_1	$Kj \setminus j$	*1*	*2*	*3*	Fy_1	$Kj \setminus j$	*1*	*2*	*3*
	1	2	0	*0*		1	1	0	*0*
	2	2	*0*	2		2	1	*0*	0
	3	0	4	0		3	0	2	0

Seekord pole Fx- ja Fy-s võrdseid sagedusi (ja seega ka reegleid). Kõik kasutatavad (mitte nullised) sagedused Fy-s on jällegi samal tunnusel (1.1 ja 1.2), seetõttu pole mõtet teha väljavõttu neist kummagi järgi. Algoritm tagurdab jälle algtasemele tagasi, kus samuti on kõik mitte nullised sagedused (Fy-s) samas veerus:

Fx_0	$K_j \setminus j$	*1*	*2*	*3*	Fy_0	$K_j \setminus j$	*1*	*2*	*3*
	1	3	3	*0*		1	1	0	*0*
	2	5	*0*	3		2	2	*0*	0
	3	0	*0*	0		3	0	*0*	0

Siinkohal lõpetab algoritm töö.

Töö käigus leiti 2 reeglit:

- R1: 2.2=1 (*Hair.red*→*Class*.+)
- R2: 3.1&2.3=2 (*Eyes.blue* & *Hair.blond*→*Class*.+)

Kasutatud näite korral me liigseid reegleid ei leidnud. Pikem näide, mille korral leitakse ka liiaseid reegleid, on toodud artiklis (Kuusik & Lind 2011).

15 Nullfaktorite probleem

Nullfaktoriteks nimetatakse faktoreid, mille panus täpsusse on null st nende lisamine või eemaldamine ei muuda reegli täpsust. Nullfaktorid reegli vasakus pooles teevad reegli pikemaks kui minimaalselt võimalik ja võivad ka suurendada reeglite arvu.

Kahjuks ei saa nullfaktorit tuvastada n-ö käigu pealt, leides reeglisse lisatava faktori panuse täpsusse lisamise hetkel ja kasutades vaid positiivseid faktoreid (positiivse panusega faktoreid). Nimelt võib juhtuda, et faktor, mis oli positiivne lisamise hetkel, pole seda enam pärast järgmiste faktorite lisamist reeglisse. Toome näite (tuttavate andmete peal):

- *Eyes.blue* → *Class*.+ (A=3/5)
- *Eyes.blue&Height.tall* → *Class*.+ (A=2/3) ΔA(*Height.tall*) = 2/3-3/5 = 1/15 >0
- *Eyes.blue&Height.tall&Hair.blond* → *Class*.+ (A=1) ΔA(*Hair.blond*) = 1-2/3 = 1/3 >0

Mõlemad reeglisse lisatavad faktorid – *Height.tall* ja *Hair.blond* – on positiivsed lisamise hetkel. Sellele vaatamata osutub, et lõplikus reeglis (*Eyes.blue&Height.tall&Hair.blond*→*Class*.+) on *Height.tall* nullfaktor (ja seega liigne) – reegel on täpne ka ilma selle faktorita:

- *Eyes.blue&Hair.blond* → *Class*.+ (A=1)

Seega ei saa me lisatava faktori positiivsust lõplikus reeglis kindlaks teha faktori panuse (täpsusse) järgi lisamise hetkel. Selline tähelepanek kehtib nii aditiivsete kui mitte-aditiivsete reeglisüsteemide korral.

Seda probleemi on kajastatud artiklis (Lind & Kuusik 2008b).

15.1 Nullfaktorite tüübid

Nagu juba öeldud, otsime reegleid, mis ei sisaldaks nullfaktoreid.

Nullfaktoreid on kaht tüüpi.

1. Nullfaktorid, mille panus täielikkusse on null (ΔC=0) – ei muuda reegli poolt kaetavate objektide hulka ja seega ka reegli sagedust (mis näitab kaetud objektide arvu) ning
2. Nullfaktorid, mille panus täielikkusse on negatiivne (ΔC<0) – vähendavad reegli sagedust (kaetud objektide arvu).

Esimesi nimetame null-nullfaktoriteks ja teisi null-negatiivseteks faktoriteks.
Näiteks reeglis

- *Height.tall&Hair.red* → *Class*.+ (A=1, C = 1/3)

on *Height.tall* null-nullfaktor, sest reegel

- *Hair.red* → *Class*.+ (A=1, C = 1/3)

katab täpselt samasid objekte ning on sama täpne ja sama täielik – seega ΔA=0 ja ΔC=0.
Reeglis

- *Height.tall&Hair.blond&Eyes.blue* → *Class*.+ (A=1, C = 1/3)

on *Height.tall* null-negatiivne faktor, sest reegel

- *Hair.blond&Eyes.blue* → *Class*.+ (A=1, C = 2/3)

on küll sama täpne (ΔA=0), kuid katab rohkem objekte: ΔC=1/3–2/3=–1/3.

Nullfaktorite tüübid on kajastatud artiklis (Lind & Kuusik 2016).

16 DA reeglite seosed suletud hulkade ja generaatoritega

Siinkohal esitame teoreetilise vaate, millele tugineb meie käsitlus mitte-liiaste (null-faktori vabade) reeglite leidmisel. Selleks defineerime mõned sagedaste elemendihulkade leidmise (*frequent itemset mining*) valdkonna mõisted, eesmärgiga kasutada neid mitte-liiaste reeglite defineerimiseks.

Sagedaste elemendihulkade leidmisel on **elemendiks** binaarne tunnus, mis kas esineb või ei esine transaktsioonis (andmebaasi kirjes). Näiteks ostukorvi andmebaasis on elementideks ostetud kaubad. Laiendades elemendi mõistet rohkemate väärtustega tunnustele, on elemendiks tunnus koos konkreetse väärtusega selle tunnuse võimalike väärtuste hulgast. Kui ostukorvi andmetes on meil nt „šokolaad", siis siinkohal võib olla kas „must šokolaad" või „valge šokolaad" (st tunnus „šokolaad" emma-kumma, s.o teineteist välistava väärtusega). Selline element vastab DA faktorile.

Suletud (elemendi)hulk (*closed (item)set*) on objektide hulga ühiste elementide maksimaalne hulk (Pasquier et al 1998), millel ei ole sagedusega ülemhulka (Zaki & Hsiao 2002). Mistahes elemendi lisamine suletud hulgale vähendab selle katet ja sagedust. Nt Quinlani tabelis (lk 66) on üheks suletud hulgaks *Hair.blond*&*Eyes.blue*&*Class*.+ sagedusega 2. Lisades sellesse hulka kas *Height.short* või *Height.tall*, muutub hulga sagedus ja saadud elemendihulk pole enam seesama suletud hulk.

Elemendihulga **sulundiks (*closure*)** on väikseim suletud hulk, mis seda elemendihulka sisaldab (Bastide et al. 2000b), st elemendihulga suurim sama sagedusega ülemhulk. Näiteks, hulga *Hair.red* (sagedusega 1) sulundiks on *Height.tall*&*Hair.red*&*Eyes.blue*& *Class*.+ (sagedusega 1). Suletud hulga sulundiks on seesama suletud hulk.

Suletud hulga **(minimaalne) generaator** on elemendihulk, millel on sama sulund ja puuduvad sama sulundiga alamhulgad (Bastide et al. 2000a). Mistahes elemendi eemaldamine minimaalsest generaatorist suurendab hulga katet ja sagedust. Näiteks, *Hair.blond*&*Eyes.blue* (sagedusega 2) on suletud hulga *Hair.blond*&*Eyes.blue*&*Class*.+ (sagedusega 2) generaatoriks. Kui eemaldaksime generaatorist kas *Hair.blond* või *Eyes.blue*, saaksime suurema sageduse ja erineva sulundiga elemendihulga.

Suletud hulgal võib olla rohkem kui üks minimaalne generaator. Näiteks, suletud hulgal *Hair.blond*&*Eyes.blue*&*Class*.+ on kaks (minimaalset) generaatorit: *Hair.blond*&*Eyes.blue* ja *Hair.blond*&*Class*.+.

Elemendihulk (nt suletud hulk või generaator) on **sagedane**, kui selle sagedus on suurem või võrdne etteantud lävega. Kui sageduslävi on 2, siis *Height.tall&Hair.red&Eyes.blue&Class*.+ ja selle generaatorid (sagedusega 1) on mittesagedased (*infrequent*); *Hair.blond&Eyes.blue&Class*.+ ja selle generaatorid (sagedusega 2) on aga sagedased.

Nüüd saame näidata meie käsitluse reeglite seoseid tutvustatud mõistetega.

Suletud hulk on objektihulga ühiste elementide maksimaalne hulk ja generaator on ühiste elementide minimaalne hulk. Suletud hulga ja generaatori vahele jäävad sellised elemendid, mille lisamine või eemaldamine ei muuda katet (kaetud objektide hulka) ja (elemendihulga) sagedust. Sellised elemendid sarnanevad DA null-nullfaktoritega, mis ei muuda DA reegli täpsust ega täielikkust. (Suletud hulkade puhul klassikuuluvust tavaliselt ei jälgita.) Järelikult, null-nullfaktoritest hoidumiseks peab reegli vasak pool olema minimaalne generaator.

Minimaalsed generaatorid ei sisalda null-nullfaktoreid, kuid võivad sisaldada null-negatiivseid faktoreid. Näiteks, generator *Height.tall&Hair.blond&Eyes.blue* determineerib klassi *Class*.+ (*Height.tall&Hair.blond&Eyes.blue→Class*.+), kuid *Height.tall* on null-negatiivne faktor, sest *Hair.blond&Eyes.blue* on piisav *Class*.+ determineerimiseks (*Hair.blond&Eyes.blue→Class*.+). *Height.tall* vähendab reegli täielikkust 1/3 võrra (2/3-lt 1/3-le). Seega, kui generaator annab reegli (generaator→klass), siis reeglid, mille vasakul poolel on selle generaatori ülemgeneraatorid, sisaldavad null-negatiivseid faktoreid ja on liigsed.

Niisiis, klassi tuvastamiseks vajame selliseid generaatoreid, mis määravad klassi ja millel pole samal ajal ühtki alamhulka, mis määraks klassi.

Seda teemat on kajastatud artiklis (Lind & Kuusik 2016).

17 Nullfaktorivaba determinatsioonanalüüs

Minimaalsetest generaatoritest, mis määravad klassi, saame moodustada reeglid minimaalne-generaator→klass (IF minimaalne-generaator THEN klass). Sel juhul saame reeglid, mille vasak pool on vaba nullfaktoritest, seetõttu nimetame oma lähenemist **nullfaktorivabaks** determinatsioon-analüüsiks.

Siinjuures on oluline teada alljärgnevat:

1) Null-nullfaktorid, mis tuleb reegli vasakust poolest välja jätta, saame viia reegli paremale poolele – järeldusse:
minimaalne-generaator→nullfaktorid (IF minimaalne-generaator THEN nullfaktorid). Sel viisil kasutatuna näitavad need nullfaktorid kaasnemist teiste faktoritega. Näiteks, *Height.tall&Hair.red* sisaldab null-nullfaktorit *Height.tall*. Viies selle faktori vasakult poolelt paremale, saame täpse reegli *Hair.red→Height.tall*:

A(Hair.red→Height.tall) = n(Hair.red&Height.tall) / n(Hair.red) = 1/1 =1.

See reegel ütleb: kellel on punased juuksed, see on ka pikka kasvu (muidugi, nii väikese sagedusega reegel nagu siin, ei ole kuigi veenev). Tegemist on täpse assotsiatsioonireegliga.

2) *Hair.red* korral on veel faktoreid, mida saame viia reegli paremale poolele: *Hair.red→Height.tall& Eyes.blue&Class*.+. Siit paistab välja, et klassi (*Class*.+) saame kindlaks teha samamoodi kui teised null-nullfaktorid. Erinevus on selles, et klassitunnust ei panda kunagi reegli vasakule poolele, samas kui ülejäänud tunnused võivad esineda emmal-kummal poolel.

3) Null-negatiivseid faktoreid ei saa vasakult paremale viia! Täpse reegli vasakuks pooleks olev *Height.short&Hair.blond&Eyes.blue* sisaldab null-negatiivset faktorit *Height.short*. Kui viime selle faktori reegli paremale poolele, saame uue reegli *Hair.blond&Eyes.blue →Height.short*, mis pole täpne:

A = n(Hair.blond&Eyes.blue&Height.short) / n(Hair.blond&Eyes.blue) = 1/2.

Edaspidi mõtleme „nullfaktori“ all null-nullfaktorit.

4) Lisaks sellele, et reegli paremal poolel võib olla klass või nullfaktor, saame leida ka reegleid, mille paremal poolel on eitus. Eitav reegel näitab, milliseid faktoreid ei sisaldu nendes objektides, mida reegli vasak pool katab. Selliseid faktoreid nimetame **välistatud faktoriteks**. Eitav reegel on kujul: minimaalne-generaator→NOT välistatud-faktor (IF minimaalne-generaator THEN NOT välistatud-faktor). Näiteks, *Eyes.brown*→NOT *Hair.red*. Kui välistatud faktoreid on rohkem kui üks (sama

generaatori korral), siis eitatakse neid ükshaaval: (NOT välistatud-faktor1) AND (NOT välistatud-faktor2) [AND …].

Välistatud faktoreid leitakse ainult nendest tunnustest, mis ei esine sama reegli üheski teises faktoris. Kui meil oleks reegel *Hair.red→Eyes.blue*, siis ei tekitataks reeglit *Hair.red*→NOT *Eyes.brown* (ja *Hair.red*→NOT *Eyes.green* – juhul, kui (andmetes) esineks ka roheliste silmadega isikuid), sest saame selletagi järeldada, et punaste juustega isikutel ei ole pruunid (ega rohelised) silmad, kuna nende silmad on sinised.

Sama reegli välistatud faktorite hulgas võib olla mitu samal tunnusel baseeruvat faktorit (samas kui eitamata faktorite korral reegli paremal poolel saab iga tunnus osaleda vaid ühe väärtusega). Meie väike tabel ei sisalda sobivat näidet. Seepärast oletame, et võimalikke silmavärve on neli: sinine (*blue*), pruun (*brown*), roheline (*green*) ja hall (*grey*). Sellisel juhul on võimalik, et punaste juustega (*Hair.red*) isikute silmad on kas sinised või hallid. Niisugusel juhul moodustataks eitavad reeglid ülejäänud silmavärvide kohta: *Hair.red* → NOT *Eyes.brown* AND NOT *Eyes.green*. Kui tunnusel on alla kolme erineva võimaliku väärtuse, siis ei saa see tunnus esineda välistatud faktorite hulgas.

Kokkuvõtvalt, nullfaktorivaba determinatsioonanalüüs leiab kolme sorti reegleid, millede vasakul poolel on minimaalne generaator:

1. Klassifikatsioonireegel:
   minimaalne-generaator→klass (IF minimaalne-generaator THEN klass)
2. (positiivne) assotsiatsioonireegel:
   minimaalne-generaator→nullfaktorid (IF minimaalne-generaator THEN nullfaktorid)
3. Negatiivne assotsiatsioonireegel:
   minimaalne-generaator→NOT välistatud-faktor
   (IF minimaalne-generaator THEN NOT välistatud-faktor)

Sama generaatori korral võime kõik kolm tüüpi ühendada:

minimaalne-generaator→klass [AND nullfaktor(id)] [AND NOT välistatud-faktor(id)]

kusjuures nullfaktorid ühendatakse konjunktsiooniga (AND) ning välistatud faktorid eitatakse ükshaaval enne nende ühendamist konjunktsiooniga: (NOT välistatud-faktor1) AND (NOT välistatud-faktor2) [AND …].

Nullfaktorivaba determinatsioonanalüüsi käsitleme osaliselt töös (Lind & Kuusik 2016), mis kajastab kaht esimest reeglitüüpi (kolmest). Kõik kolm reeglitüüpi on kajastatud Liisa Jõgiste magistritöös (2014).

Algoritm

Järgnev algoritm nullfaktorivabade reeglite leidmiseks moodustab kõiki kolme tüüpi reegleid. Algoritm põhineb generaatorite leidmisel. Iga generaatori puhul on võimalik kindlaks teha selle erinevus oma suletud hulgast s.o null-nullfaktorid, sh klass (kui on) ning leida välistatud faktorid (tunnustest, mis ei osale selles suletud hulgas). Kõigi vajalike generaatorite leidmine on garanteeritud. Nende soovimatutest ülemgeneraatoritest (mis sisaldavad nullfaktoreid) suudetakse enamikku vältida, ülejäänud eemaldatakse tulemuse kompresseerimisega (pärast põhialgoritmi).

Minimaalsete generaatorite ülemgeneraatorid loetakse liigseteks ainult klassifikatsioonireeglite korral (esimene reeglitüüp). Nii positiivsete kui negatiivsete assotsiatsioonireeglite (tüübid 2 ja 3) korral määrab reeglite sügavuse sageduslävi (minimaalne lubatud sagedus). Samuti ei minda sügavamale pärast klassifikatsioonireegli leidmist (kui ka sageduslävi lubaks seda). Näiteks, kui on leitud reegel *Eyes.brown→Class*.–(AND NOT *Hair.red*), siis enam ei uurita, kas *Eyes.brown* ülemgeneraatorite *(Eyes.brown&Hair.dark, Eyes.brown&Hair.blond*, *Eyes.brown& Height.tall*, *Eyes.brown&Height.short*) jaoks leidub nullfaktoreid ja/või välistatud faktoreid. Tegelikult leiduvad nt *Eyes.brown&Hair.dark* jaoks nullfaktorid *Height.tall* ja *Class*.– st *Eyes.brown&Hair.dark→ Height.tall&Class*.–.

Tegemist on sügavuti otsingu algoritmiga, mis teeb järjestikuseid väljavõtte kindlate faktorite järgi. Tippude sagedused vähenevad suunaga juurest lehtede poole. Iga väljavõtt on määratletud generaatoriga. Iga generaator leitakse vaid üks kord.

Algoritm kasutab sagedustabeleid, mis näitavad iga tunnuse kohta selle iga võimaliku väärtuse esinemissagedust (selles objektihulgas, mille kohta sagedustabel leitud on).

Sagedused (sagedustabelis) saavad olla võrdsed või väiksemad kui jooksev „juhtsagedus” (objektide arv jooksvas väljavõtus). Võrdne sagedus näitab, et kõik väljavõtu objektid sisaldavad seda

faktorit. Iga tunnuse kohta saab olla üks juhtsagedusega võrdne sagedus, sel juhul on selle tunnuse ülejäänud (väärtuste) sagedused nullid. Niisuguse sagedusega faktorid on null-nullfaktorid (antud väljavõtus).

Klassi kindlaks tegemine on analoogiline. Kui klassitunnuse ühe väärtuse sagedus on võrdne juhtsagedusega (ja kõik teised on nullid), siis kuuluvad kõik väljavõtu objektid sellesse klassi.

Lisaks klassile ja nullfaktoritele saab leida ka välistatud faktorid. Välistatud faktor on selline faktor, mida ei esine antud väljavõtu objektidel, kuid esineb algtabelis. Arvestatakse vaid neid tunnuseid, mis ei osale generaatoris ega nullfaktorites (s.o suletud hulgas).

Et ära hoida jooksva klassifikatsioonireegli alamreeglite (generaatori ülemgeneraatorite) leidmine, tagurdab algoritm pärast klassi kindlaks tegemist. Ära hoida saab vaid neid alamreegleid (ülemgeneraatoreid), mida pole veel leitud.

Kui klassi ei tuvastatud (väljavõtu objektid kuuluvad erinevatesse klassidesse), siis saame kontrollida, kas antud harust on võimalik veel mõnda klassifikatsioonireeglit saada. See on võimalik juhul, kui vähemalt ühe klassi sagedus on suurem või võrdne sageduslävega. Vastasel korral võib algoritm (soovi korral) tagurdada.

Kui tagurdada pole vaja, valitakse sageduse alusel (sagedustabelist) järgmine faktor generaatorisse (reegli vasakusse poolde) lisamiseks. Selle faktori sagedus peab olema väiksem jooksva väljavõtu sagedusest ja suurem või võrdne sageduslävega. Esimene tingimus välistab (antud väljavõtu) null-nullfaktorite valimise, teine aga on tavaline sagedaste hulkade ja reeglite leidmisel. Et leida ainult minimaalseid generaatoreid (mitte neid, mis jäävad minimaalse generaatori ja suletud hulga vahele), valitakse minimaalne sobiv sagedus. Kui neid on rohkem kui üks, siis lihtsalt üks neist. Valitud faktor koos (samas harus) eelnevalt valitud faktoritega moodustab generaatori ja määratleb (jooksvast) kitsama objektide hulga.

Et ära hoida varem leitud generaatorite korduv leidmine, nullitakse valitud faktori („juhtfaktori”) sagedus sagedustabelis. Enne uue juhtfaktori leidmist „tuuakse need nullid alla” eelmise taseme sagedustabelist jooksva taseme sagedustabelisse (v.a algtasemele).

Pseudokoodis kasutame järgmisi tähistusi:

- X_0 – esialgne andmetabel (objekte*tunnuseid);
- algFT – esialgne sagedustabel (väärtusi*tunnuseid);
- tarv – tunnuste arv (ilma klassita);
- klass – klassitunnus;
- t – rekursiooni tase (sügavus);
- X_t – objektide hulk (väljavõtt) tasemel t;
- FT_t – hulga X_t sagedustabel;
- V – „juhtsagedus” st väljavõtu sagedus;
- gen_t – generaator tasemel t;
- klassita – tõeväärtus, kas gen_t jaoks on klass tuvastamata;
- kl_pot – tõeväärtus, kas on võimalus leida klassireegleid järgnevatest väljavõttudest;
- gklass – gen_t klassiväärtus;
- nf_t – nullfaktorid (gen_t suhtes);
- vf – välistatud faktorid ($gen_t \cup nf_t$ suhtes);
- piir – sageduslävi (minimaalne lubatud sagedus);

faktorid on esitatud kujul $väärtus_{tunnus}$;

omistused on tähistatud „←”-ga („=” on võrdluste jaoks).

Algoritm minimaalsete generaatorite leidmiseks koos klassi, nullfaktorite ja välistatud faktoritega

Antud: X_0 , piir >0

A1. $t \leftarrow 0$; $gen_0 \leftarrow \{\}$; $nf_0 \leftarrow \{\}$

A2. leida FT_0

A3. $algFT \leftarrow FT_0$

A4. FOR EACH faktor $h_{f=1,\ldots,tarv} \in FT_0$ sagedusega V=min $FT_0[h_f] \geq piir$ DO

A5. $FT_0[h_f] \leftarrow 0$

A6. tee_väljavõtt(t+1; h_f; V)
NEXT
Algoritmi lõpp

PROCEDURE tee_väljavõtt(t; h_f; V)

B1. $gen_t \leftarrow gen_{t-1} \cup h_f$
B2. $nf_t \leftarrow nf_{t-1}$; vf←{} ; gklass←0 ; klassita←true ; kl_pot←false
B3. eraldada alamtabel $X_t \subset X_{t-1}$ nii et $X_t=\{X_{ij} \in X_{t-1} \mid X.f=h_f\}$
B4. leida FT_t
B5. IF leidub väärtus klv nii et $FT_t[klv_{klass}]$=V THEN
B6. gklass←klv ; klassita←false
B7. ELSEIF leidub väärtus klv nii et $FT_t[klv_{klass}] \geq$piir THEN
B8. kl_pot←true
ENDIF
B9. FOR EACH vaba positsioon p ($p \in 1,\ldots,tarv$) IN gen_t DO
B10. IF leidub väärtus h nii et $FT_t[h_p]$= V THEN
B11. $nf_t \leftarrow nf_t \cup h_p$
B12. ELSE
B13. FOR EACH (tunnuse p) väärtus v DO
B14. IF $FT_t[v_p]$= 0 THEN
B15. IF algFT [v_p]> 0 THEN
B16. $vf \leftarrow vf \cup v_p$
ENDIF
ENDIF
NEXT
ENDIF
NEXT
B17. väljastada gen_t, nf_t, gklass, vf, V
B18. IF V>piir AND klassita AND kl_pot THEN
B19. nullid_alla(t)
B20. FOR EACH $h_{u=1,\ldots,tarv} \in FT_t$ sagedusega V2=min $FT_t[h_u] \geq$piir and V2<V DO
B21. $FT_t[h_u] \leftarrow 0$
B22. tee_väljavõtt(t+1; h_u; V2)
NEXT
ENDIF
END PROCEDURE

PROCEDURE nullid_alla(t)

C1. FOR EACH faktor $h_{u=1,\ldots,tarv} \in FT_t$ sagedusega >0 DO
C2. IF $FT_{t-1}[h_u]$=0 THEN $FT_t[h_u] \leftarrow 0$
NEXT
END PROCEDURE

Antud on esialgne andmetabel X0 ja sageduslävi piir. Kõigepealt algväärtustatakse rekursiooni tase t, tühi generaator gen0 ja nullfaktorite tühi hulk nf0 (samm A1). Järgmiseks leitakse X0-i sagedustabel FT0 (A2) ja see algseis salvestatakse tabelisse algFT (A3). Sammul A4 valitakse (sageduste kasvavas järjestuses) iga sobiva sagedusega (≥piir) faktor juhtfaktoriks (et lisada see generaatorisse). Juhtfaktori hf sagedus nullitakse sagedustabelis FT0 (A5) ja tehakse väljavõtt hf järgi (A6).

Peaprogramm teeb väljavõtte algtabelist, kõik sügavamad väljavõtud tehakse protseduuris **tee_väljavõtt**. Protseduur alustab jooksva generaatori **gen_t** väärtustamisega (B1) ning nullfaktorite hulga **nf_t**, välistatud faktorite hulga **vf**, generaatorile vastava klassi **gklass**, indikaatori **klassita** (mis näitab, kas generaatorile on klass veel leidmata) ja indikaatori **kl_pot** (mis näitab, kas edasistest väljavõttudest saab tulla klassifikatsioonireegleid) algväärtustamisega (B2). Järgnevalt eraldatakse juhtfaktori **h_f** järgi objektide alamhulk **X_t** (B3) ja leitakse sellele vastav sagedustabel **FT_t** (B4).

Sammul B5 kontrollitakse, kas klassitunnusel **klass** leidub selline väärtus **klv**, mille sagedus on võrdne juhtsagedusega **V**. Kui leidub, siis generaator $\mathbf{gen_t}$ määrab klassi ning sammul B6 väärtustatakse sobivalt generaatorile vastav klass **gklass** ja indikaator **klassita**. Vastasel korral tehakse kindlaks, kas leidub vähemalt üks klassiväärtus sagedusega ≥**piir** (B7). Sellisel juhul on potentsiaali leida järgnevatest väljavõttudest klassireegleid ning vastavalt sellele väärtustatakse indikaator **kl_pot** (B8).

Sammul B9 läbitakse kõik jooksva generaatori $\mathbf{gen_t}$ (kui vektori) tühjad positsioonid (tunnused ilma väärtuseta) ja sammul B10 otsitakse neile väärtust sagedusega **V**. Kui selline väärtus leidub, on tegemist null-nullfaktoriga ($\mathbf{gen_t}$ suhtes) ja see faktor lisatakse nullfaktorite hulka $\mathbf{nf_t}$ (B11).

Juhul, kui positsiooni **p** jaoks pole sellise sagedusega väärtust (B12), otsime väärtust, mille sagedus oleks null (B13-B14). Neid võib esineda rohkem kui üks. Kui selline element (mille sagedus esialgses sagedustabelis **algFT** ei ole null) eksisteerib, siis on see välistatud faktor ja lisatakse välistatud faktorite hulka **vf** (B16).

Sammul B17 väljastatakse generaator koos sageduse, võimalike nullfaktorite, välistatud faktorite ja klassiga. Siinkohal saab rakendada erinevaid tingimusi otsustamaks, kas jooksev generaator väljastada või mitte – see on võimalus jätta kõrvale generaatorid, millel pole klassi ja/või nullfaktoreid (või antud tasemel uusi nullfaktoreid). Kui $\mathbf{nf_t}$ pole tühi, saame reegli $\mathbf{gen_t \rightarrow nf_t}$. Kui klass on tuvastatud, saame reegli $\mathbf{gen_t \rightarrow}$ **gklass**. Kui **vf** pole tühi, saame reegli(d) $\mathbf{gen_t \rightarrow}$NOT **vf**.

Samm B18 kotrollib, kas on mõtet teha järgmist väljavõttu. Kui sagedus **V** on suurem kui lävi **piir**, siis eksisteerib võimalus leida sagedus, mis on <**V** ja ≥**piir**. Kui klassi pole leitud (**klassita**=true), kuid on lootust seda leida järgnevatest väljavõttudest (**kl_pot**=true) pikemate generaatoritega, siis võime jätkata. Vajadusel saab need kaks viimast tingimust ära jätta.

Kui see kontroll (sammul B18) annab positiivse tulemuse, toome eelmise taseme sagedustabelist „nullid alla“ (B19). Protseduur **nullid_alla** läbib jooksva sagedustabeli ja iga nulliga mitte võrduva faktori korral (C1) kontrollib selle faktori sagedust eelmisel tasemel (C2). Kui viimane on null, saab see faktor nullise sageduse ka jooksval tasemel (C2).

Samm B20 läbib sageduste kasvavas järjestuses kõik faktorid, mis sobivad väljavõtu tegemiseks – need, mille sagedus **V2** on väiksem kui juhtsagedus **V** (et vältida null-nullfaktorite lisamine generaatorisse) ja suurem või võrdne lävega **piir**. Valitud faktori $\mathbf{h_f}$ sagedus nullitakse (B21) ja tehakse rekursiivne pöördumine protseduuri **tee_väljavõtt** poole (uue juhtfaktori $\mathbf{h_f}$ ja selle sageduse **V2**-ga).

Märkus välistatud faktorite kohta. Üldiselt päranduvad välistatud faktorid ülemiselt tasemelt alumistele (nagu klass ja teised null-nullfaktorid). Siiski on kaks põhjust, miks me koodis ei algväärtusta muutujat **vf** sama muutuja väärtusega eelmiselt tasemelt (nagu me teeme nullfaktorite hulgaga **nf**). Esiteks, kui välistatud faktorite hulgas esinev tunnus liigub suletud hulga koosseisu, ei loe me seda enam välistatud faktoriks. Teiseks, tunnusel võib olla rohkem kui üks välistatud faktor ja nende arv võib suureneda igal sügavamal tasemel, seetõttu peame selle tunnuse väärtusi niikuinii kontrollima.

Pärast peaprogrammi toimub leitud klassifikatsioonireeglite kompresseerimine. DRH-kompressioon sobib vaid reeglitele, millel on ühesugune parem pool. Seepärast me assotsiatsioonireegleid (mille paremal poolel on nullfaktorid või välistatud faktorid) ei kompresseeri.

Näide

Seekord kasutame pisut suuremat andmetabelit (Quinlan 1986), et kõikvõimalikke tekkivaid olukordi algoritmi töös avada. Originaalandmete kodeering on järgmine:

Tunnus	*Outlook*	*Temperature*	*Humidity*	*Windy*	*Class*
Kood	***Ou***	***Te***	***Hu***	***Wi***	***Cl***
1	*sunny*	*cool*	*high*	*true*	P
2	*overcast*	*mild*	*normal*	*false*	N
3	*rain*	*hot*			

Kodeeritud algandmed:

obj	*Ou*	*Te*	*Hu*	*Wi*	*Cl*
1.	1	3	1	2	2
2.	1	3	1	1	2
3.	2	3	1	2	1
4.	3	2	1	2	1
5.	3	1	2	2	1
6.	3	1	2	1	2
7.	2	1	2	1	1
8.	1	2	1	2	2
9.	1	1	2	2	1
10.	3	2	2	2	1
11.	1	2	2	1	1
12.	2	2	1	1	1
13.	2	3	2	2	1
14.	3	2	1	1	2

14 objekti on kirjeldatud 4 tunnusega ja jagatud kahte klassi.

Demonstreerime algoritmi tööd, võttes sageduslävekseks 2. Näites me ei demonstreeri algoritmi tööd algusest lõpuni (kõikide reeglite leidmiseni), vaid piirdume kahe esimese väljavõtuga algtasemelt.

Kõigepealt leiame esialgse sagedustabeli:

FT_0	*Ou*	*Te*	*Hu*	*Wi*	*Cl*
1	5	4	7	6	9
2	4→0	6	7	8	5
3	5	4	0	0	0

Siit valime esimese minimaalse sobiva sagedusega (≥lävi) faktori: Ou.2=4 (s.o tunnus Ou väärtusega 2, mille sagedus on 4). Nullime selle faktori sageduse FT_0-s (mida tähistab „→0"). Teeme väljavõtu Ou.2 järgi:

G1	2				=**4**
obj	*Ou*	*Te*	*Hu*	*Wi*	*Cl*
3.	2	3	1	2	1
7.	2	1	2	1	1
12.	2	2	1	1	1
13.	2	3	2	2	1

Esimesel real on näidatud generaator Ou.2 (tähistusega G1) ja paremal sagedus (4). Järgmiseks leiame väljavõtule vastava sagedustabeli:

FT_1	*Ou*	*Te*	*Hu*	*Wi*	*Cl*
1		1	2	2	**4**
2		1	2	2	0
3		2	0	0	0

Generaatoris G1 on kolm tühja positsiooni: tunnused Te, Hu ja Wi. Ühelgi neist kolmest tunnusest pole sagedustabelis ühegi väärtuse sagedus võrdne juhtsagedusega (=4), seega pole G1 jaoks null-nullfaktoreid. Pole ka välistatud faktoreid, sest kõik nullid FT_1-s on samades kohtades kui FT_0-s (Hu.3 ja Wi.3) – selliseid faktoreid polegi (algandmetes). Klassitunnuse veerus on väärtuse 1 sageduseks 4 (ja teistel väärtustel nullid), seega määrab generaator G1 klassi (Ou.2→Cl.1). Pärast klassi leidmist tagurdab algoritm eelmisele tasemele, et ära hoida leitud reegli alamreeglite (nt Ou.2&Te.3→Cl.1) leidmine.

Valime sagedustabelist FT_0 järgmise (minimaalse sagedusega) juhttipu: Te.1=4. Väljavõtt Te.1 (G2) järgi ja sellele vastav sagedustabel:

G2		1			=4
obj	*Ou*	*Te*	*Hu*	*Wi*	*Cl*
5.	3	1	2	2	*1*
6.	3	1	2	1	*2*
7.	2	1	2	1	*1*
9.	1	1	2	2	*1*
FT_1					
1	1		0	2	3
2	1 ↓0		**4**	2	1
3	2→0		0	0	0

Seekord on generaatoris (G2) tühjad positsioonid tunnustel Ou, Hu ja Wi. Tunnusel Hu leidub väärtus, mille sagedus on võrdne juhtsagedusega: Hu.2=4. See faktor on generaatori G2 suhtes null-nullfaktor ning siit saame reegli (Te.1→Hu.2). Sama tunnuse teisi väärtusi me välistatud faktoritena ei arvesta, kuigi nende sagedused on nullid. Klassiveerus pole sagedust 4, seega G2 ei määra klassi.

Klassi ei leitud, juhtsagedus (=4) on suurem kui sageduslävi ja klassiveerus leidub lävest suurem sagedus (Cl.1=3) – seega eksisteerib võimalus leida klassifikatsioonireegel[2] (Cl.1 jaoks). Töö jätkub „nullide alla toomisega" eelmise taseme sagedustabelist (FT_0) jooksvale tasemele. Tasemel 0 on nullitud faktori Ou.2 sagedus (mis näitab, et seda faktorit on kasutatud väljavõtu tegemiseks), see null kantakse nüüd jooksvasse sagedustabelisse FT_1 (mida tähistab „↓0" vastavas lahtris), et ära hoida võimalik edasine väljavõtt selle faktori järgi.

Sagedustabelis FT_1 leiduvad mõned „sobivad" sagedused – väiksemad kui juhtsagedus ja suuremad või võrdsed lävega (s.o < 4 ja ≥ 2). Neist valitakse esimene (minimaalne) Ou.3=2, see lisatakse generaatorisse ja tehakse väljavõtt selle faktori järgi. Jooksval tasemel selle faktori sagedus nullitakse („→0" FT_1-s). Uus generaator on Te.1&Ou.3=2 (G3). Sellele vastav väljavõtt ja sagedustabel:

G3	3	1			=2
obj	*Ou*	*Te*	*Hu*	*Wi*	*Cl*
5.	3	1	2	2	1
6.	3	1	2	1	2
FT_2					
1			0	1	1
2			**2**	1	1
3			0	0	0

Sagedustabelist me ei leia klassi ega uusi null-nullfaktoreid (peale Hu.2, mis on pärit eelmiselt tasemelt). Saab väljastada reegli Te.1&Ou.3→Hu.2, mis on eelmisel tasemel leitud assotsiatsioonireegli Te.1→Hu.2 alamreegliks. Kui sellist reeglit väljundisse ei soovita, tuleb jooksvat nullfaktorite hulka võrrelda eelmise taseme omaga. Kui erinevusi pole, jätta väljastamata. (Algoritmi pseudokoodis pole seda kontrolli kirjas, see võiks sisalduda väljastusprotseduuris.)

[2] Kui klasse on 2 ja sageduslävi=2, siis puudub tegelik vajadus sellise kontrolli järele. Kui juhtsagedus on 2, tagurdab programm (real B18) selle tõttu. Suurema juhtsageduse korral: kui klassi pole leitud (**klassita**=*true*), siis üks väärtustest on kindlasti ≥ **piir** (kui **V**=3, võivad klasside sagedused jaguneda 2+1; kui **V**=4, siis 3+1 või 2+2).

Juhtsagedus on võrdne lävega (=2) (ja seega ei saa olla ühtki sagedust, mis oleks <2 ja ≥2), seetõttu algoritm tagurdab.

Sagedustabelist FT_1 (G2) valitakse järgmine sobiv faktor: Wi.1=2. Väljavõtt ja sagedustabel:

G4		1		1	=2
obj	*Ou*	*Te*	*Hu*	*Wi*	*Cl*
6.	3	1	2	1	2
7.	2	1	2	1	1
FT_2					
1	0		0		1
2	1		**2**		1
3	1		0		0

Leiame reegli „päritud" nullfaktoriga: Te.1&Wi.1→Hu.2. Jällegi võime kontrolli rakendamisel selle väljastamata jätta. Seekord leidub ka välistatud faktor (Ou.1=0), mis annab reegli Te.1&Wi.1→NOT Ou.1. Algoritm tagurdab taas juhtsageduse tõttu.

Järgmine valik sagedustabelist FT_1 (G2) on Wi.2=2:

G5		1		2	=**2**
obj	*Ou*	*Te*	*Hu*	*Wi*	*Cl*
5.	3	1	2	2	1
9.	1	1	2	2	1
FT_2					
1	1		0		**2**
2	0		**2**		0
3	1		0		0

Seekord leiame lisaks päritud nullfaktorile nii klassi kui välistatud faktori: Te.1&Wi.2→Hu.2 AND Cl.1 AND NOT Ou.2. Nüüd on tagurdamiseks kaks põhjust: leiti klass ja juhtsagedus on liiga väike edasiste väljavõttude tegemiseks.

Eelmisel tasemel (G2) on kõik sobiva sagedusega faktorid (Ou.3=2, Wi.1=2, Wi.2=2) juba kasutatud. Nullfaktor (Hu.2=4) ei sobi väljavõtu tegemiseks.

Algoritm tagurdab algtasemele:

FT_0	*Ou*	*Te*	*Hu*	*Wi*	*Cl*
1	5	4→0	7	6	9
2	4→0	6	7	8	5
3	5	4	0	0	0

Siin on kaks faktorit „nullitud" sagedustega (Ou.2 and Te.1). Edaspidise töö käigus neid generaatorite koosseisu enam ei lisata.

Töö jätkub samamoodi. Töö käigus leitakse 39 generaatorit koos klassi, null-nullfaktorite ja välistatud faktoritega (kui neid on). Siinkohal pole vahele/väljastamata jäetud neid generaatoreid, millel on ülemgeneraatoriga samad nullfaktorid (G3, G4) või samad välistatud faktorid (G8, G9, G10). Järgnevas tabelis on toodud kõik 39 generaatorit koos sageduse, klassi, nullfaktorite ja välistatud faktoritega.

	Minimaalne generaator	**Sagedus**	**Klass**	**Null-faktorid**	**Välistatud faktorid**
G1	Ou.2	4	1		
G2	Te.1	4		Hu.2	
G3	Te.1&Ou.3	2		Hu.2	

Kui lävi on kõrgem, siis võib sellel kontrollil mõju olla. Kui piir=3 ja juhtsagedus V=4, siis on võimalus, et kõigi klasside sagedused on alla läve (2+2).

	Minimaalne generaator	Sagedus	Klass	Null-faktorid	Välistatud faktorid
G4	Te.1&Wi.1	2		Hu.2	Ou.1
G5	Te.1&Wi.2	2	1	Hu.2	Ou.2
G6	Te.3	4			Ou.3
G7	Te.3&Ou.1	2	2	Hu.1	
G8	Te.3&Hu.1	3			Ou.3
G9	Te.3&Hu.1&Wi.2	2			Ou.3
G10	Te.3&Wi.2	3			Ou.3
G11	Ou.1	5			
G12	Ou.1&Te.2	2			
G13	Ou.1&Hu.2	2	1		Te.3
G14	Ou.1&Wi.1	2			Te.1
G15	Ou.1&Hu.1	3	2		Te.1
G16	Ou.1&Wi.2	3			
G17	Ou.3	5			Te.3
G18	Ou.3&Hu.1	2		Te.2	
G19	Ou.3&Wi.1	2	2		Te.3
G20	Ou.3&Te.2	3			
G21	Ou.3&Te.2&Wi.2	2	1		
G22	Ou.3&Hu.2	3			Te.3
G23	Ou.3&Hu.2&Wi.2	2	1		Te.3
G24	Ou.3&Wi.2	3	1		Te.3
G25	Te.2	6			
G26	Te.2&Hu.2	2	1		Ou.2
G27	Te.2&Wi.1	3			
G28	Te.2&Wi.1&Hu.1	2			Ou.1
G29	Te.2&Wi.2	3			Ou.2
G30	Te.2&Wi.2&Hu.1	2			Ou.2
G31	Te.2&Hu.1	4			
G32	Wi.1	6			
G33	Wi.1&Hu.1	3			Te.1
G34	Wi.1&Hu.2	3			Te.3
G35	Hu.1	7			Te.1
G36	Hu.1&Wi.2	4			Te.1
G37	Hu.2	7			
G38	Hu.2&Wi.2	4	1		
G39	Wi.2	8			

Tulemus sisaldab 11 klassiga generaatorit (millest saab reeglid kujul generaator→klass), 6 generaatorit nullfaktoritega (reeglid generaator→nullfaktorid) ja 22 generaatorit välistatud faktoritega (reeglid generaator→NOT välistatud-faktor). Generaatoril G5 leiduvad kõik kolm eri tüüpi järeldust; mõnedel generaatoritel on neid kaht sorti. 10 generaatorit on ilma ühegi järelduseta. Sellised saab soovi korral väljastamata jätta. Samuti saab väljastamata jätta (või välja filtreerida) generaatorid, millel pole klassi või pole nullfaktoreid või välistatud faktoreid.

Leidsime 8 generaatorit klassiga 1 (Cl.1): G1, G5, G13, G21, G23, G24, G26 and G38. Kaks neist on teiste generaatorite ülemgeneraatoriteks:

- G21 (Ou.3&Te.2&Wi.2=2) on G24 (Ou.3&Wi.2=3) ülemgeneraator;
- G23 (Ou.3&Hu.2&Wi.2=2) on G24 (Ou.3&Wi.2=3) ja G38 (Hu.2&Wi.2=4) ülemgeneraator.

Faktori Te.2 lisamine generaatorisse G24 põhjustab sageduse vähenemise (3-lt 2-le), seega on Te.2 null-negatiivne faktor G21-s. Samuti vähendab Hu.2 lisamine G24-sse sagedust 1 võrra; Ou.3 lisamine

G38-sse vähendab sagedust 2 võrra (4-2). Mõlemad on null-negatiivsed faktorid generaatoris G23, kuid mitte üheaegselt.

Selliste liigsete generaatorite (mis sisaldavad nullfaktoreid) eemaldamiseks kompresseeritakse iga klassi kohta käivaid generaatoreid (algses tulemuses). Need liigsed generaatorid on alati leitud enne kui nende alamgeneraatorid. Seda saab kompresseerimisel arvesse võtta. Samuti saab kasutada sagedusi potentsiaalsete liiaste reeglite leidmisel. Null-negatiivset faktorit sisaldavad generaatorid on väiksema sagedusega kui nende alamgeneraatorid; null-nullfaktoreid sisaldavad generaatorid aga sama sagedusega. Seega tasub kontrollida generaatoreid, mille sagedus on väiksem või võrdne lühema generaatori omast.

Pärast kompresseerimist jääb klassi Cl.1 jaoks järele 6 generaatorit. Järgnevas tabelis esitatakse need koos kõigi neist saadavate reeglitega ja koodide asemel kasutatakse originaalseid väärtusi.

	Minimaalne generaator	**Reeglid**
G1	*Outlook.overcast*	*Outlook.overcast→Class.P*
G5	*Temperature.cool &Windy.false*	*Temperature.cool&Windy.false→Class.P* *Temperature.cool&Windy.false→Humidity.normal* *Temperature.cool&Windy.false→NOT Outlook.overcast*
G13	*Outlook.sunny& Humidity.normal*	*Outlook.sunny&Humidity.normal→Class.P* *Outlook.sunny&Humidity.normal→Temperature.hot*
G24	*Outlook.rain &Windy.false*	*Outlook.rain&Windy.false→Class.P* *Outlook.rain&Windy.false→NOT Temperature.hot*
G26	*Temperature.mild &Humidity.normal*	*Temperature.mild&Humidity.normal→Class.P* *Temperature.mild&Humidity.normal→NOT Outlook.overcast*
G38	*Humidity.normal &Windy.false*	*Humidity.normal&Windy.false→Class.P*

Iga tabelis esitatud (minimaalse) generaatori jaoks saame moodustada nullfaktorivaba klassifikatsioonireegli (minimaalne-generaator→klass). Näiteks, G5-st saame järeldada: *Temperature.cool*&*Windy.false→Class*.*P*. Selline järeldus kehtib antud andmete korral.

Generaatoril G5 on ka nullfaktoreid, mis annavad assotsiatsioonireegli (minimaalne-generaator→nullfaktor(id)): *Temperature.cool*&*Windy.false→Humidity.normal*. See näitab, et tingimusega *Temperature.cool*&*Windy.false* kaasneb alati *Humidity.normal*.

Nagu juba nägime, kaasneb *Humidity.normal* (Hu.2) juba faktoriga *Temperature.cool* (Te.1) ja pärandub kõigile Te.1 ülemgeneraatoritele(alamreeglitele) (G3, G4, G5).

Generaator koos null-nullfaktoritega moodustab suletud hulga (close set) – kaetud objektide kõigi ühiste faktorite hulga. Ükski teine objekt (peale suletud hulga poolt kaetud objektide) (antud andmetes) ei sisalda kõiki neid faktoreid. Kahel objektil, mida katab generaator *Temperature.cool*&*Windy.false*, on 3 ühist faktorit: *Temperature.cool*&*Windy.false*& *Humidity.normal*.

Generaatoril G5 on ka välistatud faktoreid, seega saame negatiivse assotsiatsioonireegli (minimaalne-generaator→NOT välistatud-faktor):
Temperature.cool&*Windy.false*→NOT *Outlook.overcast*. See reegel ütleb: kui on jahe (*cool*) ja pole tuuline (*Windy.false*), siis ilm ei ole pilves (*overcast*) (antud andmete korral). Seega, saame minimaalsest generaatorist *Temperature.cool*&*Windy.false* (G5) järeldada klassi *Class*.*P* ning tingimuse *Humidity.normal* esinemise ning tingimuse *Outlook.overcast* mitteesinemise.

18 Mitmete ülesannete määratlemine kliki leidmise ülesandena

Käesolevas peatükis näitame, et eelpool kirjeldatud mitmed meetodid on oma olemuselt käsitletavad kliki leidmise ülesandena. Selleks defineerime vajalikud mõisted, määratleme iga meetodi korral selle lähteülesande ja seejärel sõnastame lähteülesande kliki leidmise ülesandena. Iga meetodi korral toome ka vastava näite.

Definitsioon 4.1. **Kahealuseline graaf** (*bipartite graph*) on graaf, mille tippude hulk koosneb kahest lõikumatust osast, nii et ühte ossa kuuluvad tipud ei ole omavahel servaga seotud.

Definitsioon 4.2. **Biklikk** (*biclique*) on klikk kahealuselises graafis.

Näide

Koosnegu kahealuseline graaf tipuhulkadest A ja B. Olgu hulgas A 4 tippu ja hulgas B 6 tippu, kusjuures mõned hulkade A ja B tipud on omavahel kaarega seotud. Sellist graafi saab kujutada kahe tipuhulga vahelise seosmaatriksina (AB), kus „1" tähistab kaare olemasolu ja „0" kaare mitte olemasolu nende kahe tipu vahel:

A/B	*B1*	*B2*	*B3*	*B4*	*B5*	*B6*
A1.	1	0	1	0	0	1
A2.	1	1	0	0	0	1
A3.	1	0	0	1	1	0
A4.	1	1	0	0	0	1

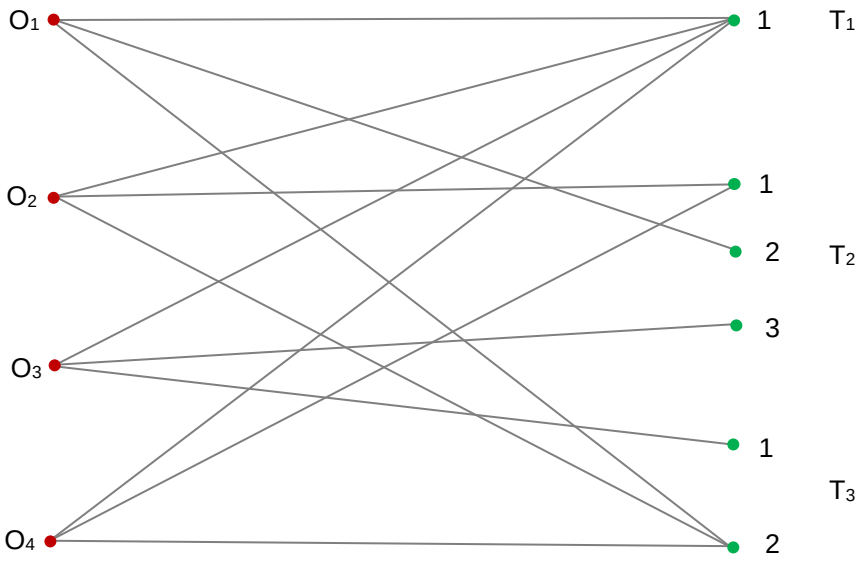
Sellest graafist saab eraldada mitmeid biklikke, näiteks:

{(kaar)A1B1, A1B6; A2B1, A2B6; A4B1, A4B6}, {A2B1, A2B2, A2B6; A4B1, A4B2; A4B6}, {A1B1, A1B3, A1B6} jt.

Selgub, et iga diskreetset objekt-tunnus andmetabelit saab esitada kahealuselise graafina, mille üheks aluseks on objektid (O, read), teiseks aluseks on tunnused (T, veerud). Vaatame näiteks järgmist objekt-tunnus tüüpi andmetabelit:

O/T	*T1*	*T2*	*T3*
O1.	1	2	2
O2.	1	1	2
O3.	1	3	1
O4.	1	1	2

Näeme, et tunnusel T1 on skaala pikkusega 1, T2 korral 1,2,3 ja T3 korral 1,2. Iga tunnuse iga skaalaväärtust saame käsitleda kahendgraafi teise aluse tippude hulgana, seega antud näites teises aluses on 1+3+2=6 tippu. Andmetabelile vastav graaf näeb välja selline:



Kui me esitaksime selle graafi kirjelduse seosmaatriksina, siis saaksime samasuguse seosmaatriksi, kui on seosmaatriks AB. Kui graafiteoreetilised algoritmid eeldavad oma töös graafi esitamist kas seosmaatriksina, naabrusmaatriksina, kaarte loeteluna või muul kujul, siis käesolevas töös esitatud algoritmid lähtuvad andmetabelist, st teisendus andmetabelist näiteks seosmaatriksiks pole vajalik.

Lähtudes eelpool kirjeldatud andmetabeli kahealuselise graafina esitamise käsitlusest saame asuda mitmete raamatus kirjeldatud meetodite määratlemist bikliki leidmise ülesandena.

18.1 Hüpoteeside generaator

Hüpoteeside generaatori ülesandeks on eraldada analüüsitavast andmetabelist kõik suletud hulgad (*closed set*). Iga suletud hulk koosneb talle vastava objektide hulga maksimaalsest kirjeldusest, so kõikides selle objektides samaaegselt esinevatest ühesugustest faktoritest (tunnus-väärtus), s.o lõige üle analüüsitava objektide hulga. Sellest johtuvalt saame HG ülesande määratleda järgmiselt:

Leida kõik (maksimaalsed) biklikid kahealuselises graafis.

Eelpool toodud andmetabeli korral eraldatakse kolm biklikki (eeldusel, et üksikut objekti ei käsitleta biklikina):
{O1T1.1, O2T1.1, O3T1.1, O4T1.1}, {O1T1.1, O1T3.2; O2T1.1, O2T3.2; O4T1.1, O4T3.2}, {O2T1.1T2.1T3.2, O4T1.1T2.1T3.2}

18.2 Determinatsioonanalüüs

Determinatsioonanalüüsi originaalmeetodi korral antakse ette teatud omadusega objektide hulk (valim, konkreetne, ette antud klassiväärtus: klass.väärtus) Y ja leitakse seda kirjeldavad reeglid. Töö lõpetatakse, kui kõik Y objektid on reeglitega kaetud. St, et ei leita mitte kõiki reegleid, vaid (soovitavalt) võimalikult vähene reeglite hulk. Nii saame DA ülesande määratleda järgmiselt.

Leida (võimalikult minimaalne) biklikkide hulk niimoodi, et kahealuselise graafi O osa tipud, mis kuuluvad hulka Y, oleksid kaetud.

Meie andmetabeli korral, kui valime objektide alamhulgaks, millele reegleid leiame, Y=T3.2, siis nende 100% katmiseks eraldatakse kaks biklikki:
{T1.1, T2.1}, {T1.1, T2.2}.

18.3 Klassireeglite leidmine

Laiendatud determinatsioonanalüüsi meetod ei ole kitsendatud Y valikuga, vaid leiab reeglid kõikidele klassitunnuse väärtustele. Seejuures tuleb lähtuda piisavuse ja täielikkuse nõudest:
Piisavus. Iga näide on kaetud ainult ühe klassi reegliga.
Täielikkus. Kõik näited kaetud vähemalt ühe reegliga.

Nii saame klassireeglite ülesande määratleda järgmiselt.

Leida biklikkide hulk kahealuselises graafis niimoodi, et oleks täidetud piisavuse ja täielikkuse tingimus.

Meie andmetabeli korral, kui valime objektide klassitunnuseks T3, millel on 2 väärtust (klassi), siis eraldatakse kolm biklikki:
{T1.1, T2.1}, {T1.1, T2.2} → klass T3.2,
{T1.1, T2.3} → klass T3.1.

Algoritmi samm-sammulist tööd demonstreerib video: MONSIL.

Kirjanduse loetelu

Bastide, Y., Pasquier, N., Taouil, R., Stumme, G., Lakhal, L. (2000). Mining minimal non-redundant association rules using frequent closed itemsets. *CL'2000 international conference on Computational Logic*, *LNCS 1861*, pp. 972–986.

Bastide, Y., Taouil, R., Pasquier, N., Stumme, G., Lakhal, L. (2000). Mining Frequent Patterns with Counting Inference. *ACM SIGKDD Explorations, 2*(2), 66–75.

DALSolution. (2007, 02 27). *DALSolution software and technology. Questions and Answers*. Retrieved from http://www.dalsolution.com/faq.htm, 27.02.2007

Jõgiste, L. (2014). *Prototyping of Zero-factor based DA.* Master's Thesis, Tallinn University of Technology, IT Faculty, Tallinn.

Kuusik, R. (1993). The super-fast algorithm of hierarchical clustering and the theory of monotone systems. *Data processing. Problems of programming. Computer science. Control engineering, Transactions of Tallinn Technical University*, pp. 37–62.

Kuusik, R., Lind, G. (2008, May). Algorithm MONSA for All Closed Sets Finding: basic concepts and new pruning techniques. *WSEAS Transactions on Information Science and Applications, 5*(5), 599-611.

Kuusik, R., Lind, G. (2010). Some Developments of Determinacy Analysis. *Advanced Data Mining and Applications: The 6th International Conference on Advanced Data Mining and Applications (ADMA2010), Chongqing, China, November 19–21, 2010. LNAI 6440*, pp. 593-602. Berlin Heidelberg: Springer-Verlag.

Kuusik, R., Lind, G. (2011). New Developments of Determinacy Analysis. In J. Tang, I. King, L. Chen, & J. Wang (Ed.), *Advanced Data Mining and Applications - 7th International Conference: ADMA 2011, Beijing, China, December 17–19, 2011. II; LNCS 7121*, p. 223−236. Springer.

Kuusik, R., Lind, G. (2012). An Effective Inductive Learning Algorithm for Extracting Rules. In F. L. Gaol, & Q. V. Nguyen (Ed.), *Proceedings of the 2011 2nd International Congress on Computer Applications and Computational Science, 2: CACS 2011, Bali, Indonesia, November 15–17, 2011. AISC 145*, pp. 339-344. Berlin Heidelberg: Springer-Verlag.

Lind, G., Kuusik, R. (2007). Some Ideas for Determinacy Analysis Realisation. *Proceedings of the 11th IASTED International Conference on Artificial Intelligence and Soft Computing. Palma de Mallorca, Spain, August 29–31, 2007* (pp. 185-190). ACTA Press.

Lind, G., Kuusik, R. (2008a, October). New developments for Determinacy Analysis: diclique-based approach. *WSEAS Transactions on Information Science and Applications, 5*(10), 1458–1469.

Lind, G., Kuusik, R. (2008b). Some Problems in Determinacy Analysis Approaches Development. *Proceedings of the 2008 International Conference on Data Mining (DMIN 2008), Las Vegas, Nevada, USA, July 14–17, 2008. Volume I*, pp. 102–108. CSREA Press.

Lind, G., Kuusik, R. (2016). Algorithm for Finding Zero Factor Free Rules. *Man-Machine Interactions 4: 4th International Conference on Man-Machine Interactions, ICMMI 2015 Kocierz Pass, Poland, October 6–9, 2015. AISC 391*, pp. 421-435. Springer.

Mullat, J. E. (1971). On the Maximum Principle for some Set Functions. *Proceedings of the Tallinn Technical University(313), 37-44. Retrieved 02 2016, from* ttp://www.datalaundering.com/download/modular.pdf

Mullat, J. E. (1976). Extremal Subsystems of Monotonic Systems. I. *Automation and Remote Control, 37(5), 758-766. Retrieved 02 2016, from* www.datalaundering.com/download/extrem01.pdf

Mullat, J. E. (1977a). Extremal Subsystems of Monotonic Systems. II. *Automation and Remote Control, 37(8), 1286-1294. Retrieved 02 2016, from* www.datalaundering.com/downlaods/extrem02.pdf

Mullat, J. E. (1977b). Extremal Subsystems of Monotonic Systems. III. *Automation and Remote Control, 38(1), 89-97. Retrieved 02 2016, from* www.datalaundering.com/downlaods/extrem03.pdf

Mullat, I., Vyhandu, L. (1979). Monotonic systems in scene analysis. In *Symposium, Mathematical Processing of Cartographic Data, Tallinn,* pp.63–66.

Pasquier, N., Bastide, Y., Taouil, R., Lakhal, L. (1998). Pruning Closed Itemset Lattices for Association Rules. *Proceedings of the BDA Conference*, (pp. 177–196).

Quinlan, J. R. (1984). Learning efficient classification procedures and their application to chess and games. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine Learning. An Artificial Intelligence Approach* (pp. 463–482). Berlin Heidelberg New York Tokyo: Springer-Verlag.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning, 11*(1), 81–106.

Zaki, M. J., Hsiao, C.-J. (2002). CHARM: An Efficient Agorithm for Closed Itemset Mining. *Proceedings of the Second SIAM International Conference on Data Mining*, *2*, pp. 457–473.

Võhandu, L.; Kuusik, R.; Torim, A.; Aab, E.; Lind, G. (2006). Some Monotone Systems Algorithms for Data Mining. *WSEAS Transactions on Information Science and applications*, pp. 802–809

Vyhandu, L. (1989). Fast Methods in Exploratory Data Analysis. *Transactions of Tallinn Technical University*, No. 705, pp. 3-13.

Выханду, Л. К. (1979). Экспрессметоды анализа данных. Труды Таллинского Политехнического Института, № 464, с. 21-37.

Выханду, Л. К. (1980). О некоторых методах упорядочения объектов и признаков в системе данных. Труды Таллинского Политехнического Института, № 482, с. 43–50.

Выханду, Л. К. (1981). Скоростные методы обработки данных. Вычислительные системы: машинные методы обнаружения закономерностей, № 88, с. 56-64. Новосибирск: Институт математики СО АН СССР.

Контекст (1998). ДА-система 4.0 Руководство пользователя Версия 1.0 ©1998, 1999 ООО Фирма "Контекст".

Контекст (1999). ДА-система 4.0, версия 4.0 для Windows 95, Windows 98 и Windows NT. Вопросы-Ответы. ДА-система и технология анализа данных. © 1999 Фирма "Контекст".

Чесноков С. В. (1980). Детерминационный анализ социально-экономических данных в режиме диалога. Препринт. Москва, Всесоюзный научно-исследовательский институт системных исследований.

Чесноков С. В. (1982). Детерминационный анализ социально-экономических данных. Москва, Наука.

Чесноков С. В. (2002). Детерминационный Анализ Социально-Экономических Данных. Иллюстративные материалы к лекциям: Лекция 2. Правила; Лекция 3. Системы правил. Московский Государственный Университет им. М. В. Ломоносова, Экономический факультет, Москва. (неопубликованный).

Juhendatud doktori- ja magistritööde osaline loetelu

Raamatu autorite poolt andmeteaduse valdkonnas juhendatud doktori- ja magistritööde osaline loetelu (1997–2017).

	Nimi	Kraad	Kaitsmise aasta	Pealkiri originaalkeeles
1.	Taivo Rist	magistrikraad (teaduskraad)	1997	Monotoonsed süsteemid lõplikest lihtgraafidest kõikide klikkide eraldamisel:ajalise keerukuse hindamine
2.	Jüri Seiger	magistrikraad (teaduskraad)	2001	Monotoonsed süsteemid disjunktiivsete normaalkujude minimeerimisel
3.	Eik Aab	magistrikraad	2002	Monotoonsed süsteemid hägusate klastrite leidmisel
4.	Veiko Jakovets	magistrikraad (teaduskraad)	2003	Monotoonsete süsteemide rakendamine masinõppes
5.	Deniss Kumlander	doktorikraad	2005	Some practical algorithms to solve the maximum clique problem
6.	Tarmo Veskioja	doktorikraad	2005	Stable marriage problem and college admission
7.	Tanel Alumäe	doktorikraad	2006	Methods for Estonian Large Vocabulary Speech Recognition
8.	Marko Morel	magistrikraad	2006	Monotoonsetel süsteemidel põhineva masinõppe algoritmi arendus
9.	Tarvo Treier	magistrikraad	2007	Eelkorrastuse rakendamine monotoonsetel süsteemidel baseeruvale masinõppele
10.	Raino Kolk	magistrikraad	2007	Tehisnärvivõrguga elektritarbimise prognoosimine
11.	Toomas Kirt	doktorikraad	2007	Concept formation in exploratory data analysis: case study of linguistic and banking data
12.	Rain Öpik	magistrikraad	2007	Piiratud mälumahuga mudel diskreetselt järjestuselt ennustamiseks eesti keele näitel
13.	Innar Liiv	doktorikraad	2008	Pattern Discovery Using Seriation and Matrix Reordering : A Unified View, Extensions and an Application to Inventory Management
14.	Karin Lindroos	doktorikraad	2008	Mapping Social Structures by Formal Non-Linear Information Processing Methods: Case Studies of Estonian Islands Environments
15.	Ants Torim	doktorikraad	2009	Formal Concepts in the Theory of Monotone Systems
16.	Kirill Tšepurov	magistrikraad	2010	Kõikide maksimaalsete klikkide leidmine: monotoonsetel süsteemidel põhineva algoritmi parendamine
17.	Kenneth Geers	doktorikraad	2011	Strategic Cyber Security: Evaluating Nation-State Cyber Attack Mitigation Strategies with DEMATEL
18.	Tanel Tiits	magistrikraad	2011	Disjunktiivsete normaalkujude minimeerimine monotoonsete süsteemide meetoditega
19.	Ottokar Tilk	magistrikraad	2012	Iteratiivsel astendamisel põhinevad lahendused mõnedele NP-keerulistele probleemidele suurtel hõredatel maatriksitel
20.	Ermo Täks	doktorikraad	2013	An Automated Legal Content Capture and Visualisation Method

21.	Liisa Jõgiste	magistrikraad	2014	Prototyping of Zero-factor based DA
22.	Meelis Pruks	magistrikraad	2014	Realization of Equivalence Class Based Clustering
23.	Martin Rebane	magistrikraad	2014	Solving rule set optimisation problem of the MONSA family of algorithms and implementation in Java
24.	Ahti Lohk	doktorikraad	2015	A System of Testpatterns to Check and Validate the Semantic Hierarchies of WordNet-type Dictionaries
25.	Kairit Sirts	doktorikraad	2015	Non-Parametric Bayesian Models for Computational Morphology.
26.	Grete Lind	doktorikraad	2017	From Determinacy Analysis to Zero Factor Free Determinacy Analysis and Universal Generator of Hypotheses: Development of Algorithms
27.	Fjodor Ševtšenko	magistrikraad	2017	Zero Factors Free Rules Algorithm: the Study of Classification Function
28.	Andre Veski	doktorikraad	2017	Agent-Based Computational Experiments in Two-Sided Matching Markets
29.	Igor Anohhin	magistrikraad	2017	Masinaõpe ning andmete otsimine pettuste avastamiseks.

Leo Võhandu, PhD (1955)
TTÜ emeriitprofessor
üks monotoonsete süsteemide teooria loojatest (1970-ndad aastad), selle teooria edukas rakendaja ja propageerija

Rein Kuusik, PhD (1989)
TTÜ emeriitprofessor, Leo Võhandu õpilane
tegelenud monotoonsete süsteemide rakendamisega alates aastast 1984

Grete Lind, PhD (2017)
Leo Võhandu ja Rein Kuusiku õpilane
tegelenud monotoonsete süsteemide rakendamisega alates aastast 1996