TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Tarkvarateaduse instituut

Marko Mets
163552IAPM

KONTSEPTIVÕRE LIHTSUSTAMINE KONTSEPTIAHELATE ABIL

Magistritöö

Juhendaja: Ants Torim, PhD

Tallinn 2018

Autorideklaratsioon

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

Autor: Marko Mets

07.05.2018

Annotatsioon

Formaalne kontseptianalüüs on andmete analüüsimise ning andmete struktuuride visualiseerimise meetod. Analüüsitav andmestik koosneb objektidest, atribuutidest ning objektide ja atribuutide vahelistest seostest. Andmestikus leitakse kontseptid, mille moodustavad objektide hulk ning atribuutide hulk. Kontseptid ning nendevahelised seosed visualiseeritakse kasutades kontseptivõresid. Mõnesaja objektiga andmestikus võib olla tuhandeid kontsepte ning kontseptivõre ei ole loetav. Kontseptivõrede loetavamaks muutmiseks on olemas mitmeid lähenemisi, kus vähendatakse kontseptide arvu, kuid mitte ükski neist lähenemistest ei ole muutunud standardiks.

Magistritöö eesmärgiks on arendada algoritm, mille abil saab kontseptivõredest leida mitu kontseptiahelat. Töö põhineb juba olemasoleval algoritmil, mille abil on võimalik leida ainult ühte kontseptiahelat. Mitme kontseptiahela leidmist analüüsitakse rakendades seda meetodit mitme erineva andmestiku peal, mis sisaldavad kümneid tuhandeid kontsepte. Analüüsi käigus leitakse kuivõrd hästi kontseptiahelad algset andmestikku katavad ning kuivõrd palju parem on mitme kontseptiahela leidmine ühe kontseptiahela leidmisest.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 55 leheküljel, 7 peatükki, 11 joonist, 5 tabelit.

Abstract

Simplifying a concept lattice with concept chains

The aim of this thesis is to improve readability of concept lattices which are used in formal concept analysis.

Formal concept analysis is a method for data analysis and data structure visualization. Dataset to be analyzed consists of objects, attributes and binary relations between objects and attributes. With the help of formal concept analysis concepts can be derived from the dataset. Concept is a pair consisting of a set of objects and a set of attributes where each object in the first set has all the attributes in the second set. Concepts have hierarchical relations which can be visualized by using concept lattices. Problem with concept lattices is that even relatively small datasets consisting of a few hundred objects can generate rather huge concept lattices with thousands of concepts. Concept lattices with such dimensions are impossible for humans to analyze visually. Many different approaches for reducing the number of concepts in the lattice exist but as of now there is no standard way of doing it.

This thesis proposes another approach for reducing the visual complexity of the concept lattice. Concept reduction is achieved by finding multiple concept chains from the concept lattice. Concept chain is a set of structurally ordered concepts from top of the concept lattice to the bottom of the concept lattice. This thesis is based on an already existing work in which finding of a single concept chain was implemented.

As a result, an algorithm for finding multiple concept chains from a dataset was developed. New method was applied on datasets that contain tens of thousands of concepts. The experiments proved that finding more than one concept chain yields a much bigger coverage of the original dataset and therefore the new approach is an improvement over the initial solution where only one concept chain was found. Analysis of the proposed method also shows that it is possible to reduce the number of concepts drastically while not losing too much information contained in the original dataset.

The thesis is in Estonian and contains 55 pages of text, 7 chapters, 11 figures, 5 tables.

Lühendite ja mõistete sõnastik

FCA	*Formal concept analysis*, formaalne kontseptianalüüs
CSV	*Comma-separated values*, komadega eraldatud väärtused

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Formaalne kontseptianalüüs on andmete analüüsi ning teadmiste esitamise meetod. Analüüsitavad andmed esitatakse kontseptidena. Kontseptid moodustavad omavahelise hierarhia, mida saab visualiseerida kasutades kontseptivõresid.

1.1 Probleem

Kontseptivõres on olemas palju kontseptiahelaid. Kontseptiahel on tee kontseptivõre esimesest kontseptist kontseptivõre viimase kontseptini. Samuti on kontseptiahelates igal kontseptil ainult üks vahetu ülem- ja alamkontsept, v.a kontseptiahela esimesel ja viimasel kontseptil. Kontseptivõred, milles on palju kontsepte, ei ole inimsilma jaoks lihtsasti loetavad, siis kontseptiahelate leidmine võimaldab märgatavalt vähendada kontseptide arvu ning aitab seeläbi analüüsitavaid andmeid paremini visualiseerida.

1.2 Eesmärk

Magistritöö eesmärgiks on tegeleda kontseptivõres esinevate kontseptiahelate leidmise algoritmi edasiarendusega. Algne algoritm, mis suudab leida korraga ainult ühe kontseptiahela, on loodud Ants Torimi poolt ning ei ole ametlikult teadusartiklina avaldatud. Edasiarenduse eesmärgiks on algoritmi optimeerida ning lisada võimalus mitme kontseptiahela leidmiseks.

Seejärel parendatud algoritm rakendatakse päris andmestike peal, et hinnata uue lahenduse headust. Edasiarendatud algoritmi headust saab hinnata selle järgi, kui suur protsent algsest andmetabelist on algoritmi poolt kaetud ning kas rohkem kui ühe kontseptiahela leidmisel muutub andmetabeli katvuse protsent suuremaks.

Töö hüpoteesiks on väide, et rohkem kui ühe kontseptiahela leidmisel on konteksti katvus suurem kui ainult ühe kontseptiahela leidmisel.

1.3 Ülevaade tööst

Teises peatükis tutvustatakse formaalse kontseptianalüüsi põhimõisteid ning näidatakse, mis on kontseptivõre ja kuidas seda lugeda.

Kolmandas peatükis tutvustatakse probleemi, mis kaasneb suure kontseptide arvuga kontseptivõrede esitamisel ning antakse ülevaade seotud töödest, milles on juba otsitud erinevaid lahendusi suurte kontseptivõrede lihtsustamiseks.

Neljandas peatükis tutvustatakse klasterdamist, monotoonseid süsteeme ning miinustehnikat, mida töö praktilises osas kasutatakse. Samuti tutvustatakse esialgset algoritmi, mida selles töös hakatakse edasi arendama. Lisaks selgitatakse, kuidas arvutada, kui suur osa kontekstist on kontseptiahela poolt kaetud. Katvuse abil on võimalik hinnata algoritmi headust.

Viiendas peatükis selgitatakse uue lahenduse ideed ning tutvustatakse igat algoritmi osa põhjalikumalt. Lisaks algoritmi seletusele rakendatakse seda väikese konteksti põhjal, et näidata algoritmi toimimist reaalse andmestiku peal.

Kuuendas peatükis analüüsitakse loodud algoritmi. Esmalt rakendatakse seda võrdlemisi väikese konteksti peal, et näidata, kuidas kontseptiahelate leidmisel konteksti suurus kasvab ning seletatakse, millest katvuse kasv on tingitud. Seejärel rakendatakse algoritmi kontseptiahelate katvuse hindamiseks erinevate suurte andmestike peal, milles on kümneid tuhandeid kontsepte. Samuti räägitakse algoritmi jõudlusest ning kuidas seda parandada. Ühtlasi võrreldakse loodud lahendust olemasoleva lahendusega.

2 Formaalne kontseptianalüüs

Formaalne kontseptianalüüs on andmeanalüüsi ning teadmiste esitamise meetod, mis loodi saksa teadlase Rudolf Wille poolt aastal 1982 [1]. Formaalse kontseptianalüüsi põhimõisteteks on formaalne kontekst ning formaalne kontsept. Formaalsus väljendub selles, et tegemist on matemaatiliste mõistetega, mis kajastavad ainult mõningaid sõnade kontekst ja kontsept tähendusi [2].

Selles peatükis antakse lühike ülevaade formaalsest kontseptianalüüsist ning selgitatakse selle valdkonna põhimõisteid.

2.1 Formaalne kontekst

Formaalseks kontekstiks nimetatakse struktuuri $K = (G, M, I)$, kus G on formaalsete objektide hulk, M on formaalsete atribuutide hulk ning I on objektide ja atribuutide vaheline binaarne suhe (objektil on atribuut). Formaalse konteksti põhjal defineeritakse formaalsed kontseptid. Formaalset konteksti esitatakse binaarse andmetabeli kujul, kus tabeli read tähistavad objekte, tabeli veerud tähistavad atribuute ning 'X' tähistab objekti ja atribuudi vahelise suhte olemasolu [3]. Formaalse konteksti näide on esitatud Tabel 1-s. Kuigi formaalne kontseptianalüüs tegeleb ainult binaarsete andmetabelitega, siis see ei tähenda, et mitme väärtusega andmetabeleid ei saaks kasutada. Selleks tuleb mitme väärtusega andmetabel teisendada binaarseks andmetabeliks teisendades ühe atribuudi mitmeks atribuudiks. Loomulikult võib sellega kaasneda mõningane informatsiooni kadu.

Tabel 1. Formaalse konteksti näide.

	Vajab eluks vett	Taim	Elab maa peal	Elab vees	Imetaja	Lind	Lendab	Liigub kahel jalal
Koer	X		X		X			
Karu	X		X		X			
Tamm	X	X	X					
Pingviin	X		X			X		X
Ahven	X			X				

Vesiroos	X	X		X				
Delfiin	X			X	X			
Kotkas	X		X			X	X	
Inimene	X		X		X			X

2.2 Formaalne kontsept

Formaalne kontsept leitakse formaalsest kontekstist ning defineeritakse kui paar (A, B), kus A on kõigi objektide G alamhulk ning B on kõigi atribuutide M alamhulk. Kõik hulga A objektid omavad kõiki atribuute hulgas B. Hulka A nimetatakse kontsepti ekstensiooniks ning hulka B nimetatakse kontsepti intentsiooniks. Ekstensioonile ning intentsioonile on rakendatav tuletuse operaator, mida tähistatakse ülakomaga ('). Rakendades tuletuse operaatorit objektihulgale A saadakse hulk atribuute, mis on ühised kõigile hulga A objektidele. Samalaadselt rakendades tuletuse operaatorit atribuutide hulgale B saadakse hulk objekte, mis on ühised kõigile hulga B atribuutidele. Matemaatilisel kujul saab seda väljendada järgnevalt:

$$A' = \{m \in M \mid (\forall g \in A),\ gIm\}$$

$$B' = \{g \in G \mid (\forall m \in B),\ gIm\}$$

Sellele reeglile toetudes saab defineerida ka vajaliku tingimuse, et paari (A, B) saaks nimetada kontseptiks: $A' = B$ ning $B' = A$. Seega objektide ning atribuutide paari (A, B) loetakse formaalseks kontseptiks ainult sel juhul, kui on täidetud järgnevad neli tingimust [3]:

1. $A \subseteq G$
2. $B \subseteq M$
3. $A' = B$
4. $B' = A$

Tabel 1-s esitatud konteksti põhjal saab leida 15 erinevat kontsepti, mis kõik vastavad eeltoodud neljale reeglile:

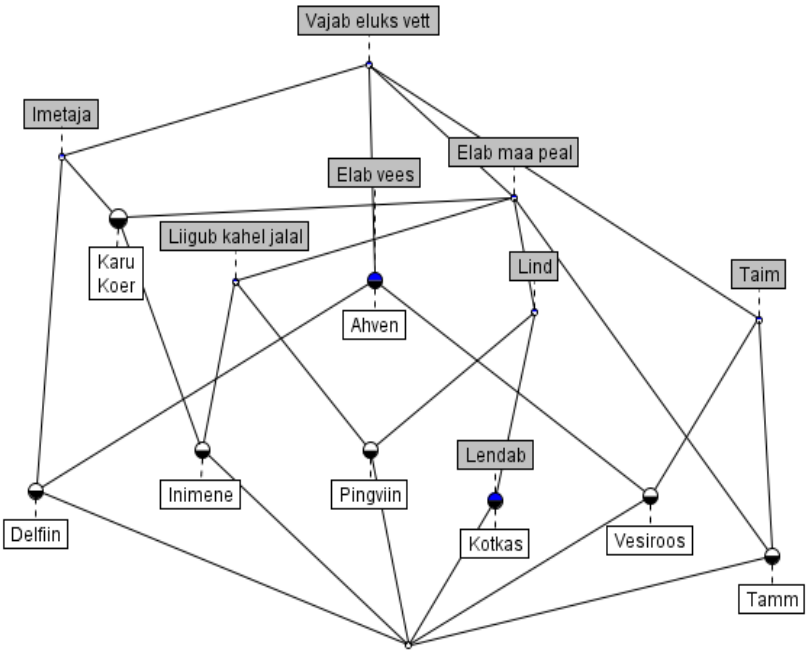
1. ({Delfiin, Koer, Karu, Ahven, Inimene, Pingviin, Kotkas, Vesiroos, Tamm}, {Vajab eluks vett})

2. ({Delfiin, Koer, Karu, Inimene}, {Vajab eluks vett, Imetaja})
3. ({Koer, Karu, Inimene}, {Vajab eluks vett, Imetaja, Elab maa peal})
4. ({Inimene}, {Vajab eluks vett, Imetaja, Elab maa peal, Liigub kahel jalal})
5. ({Ahven, Vesiroos, Delfiin}, {Vajab eluks vett, Elab vees})
6. ({Delfiin}, {Vajab eluks vett, Elab vees, Imetaja})
7. ({Vesiroos, Tamm}, {Vajab eluks vett, Taim})
8. ({Koer, Karu, Inimene, Pingviin, Tamm, Kotkas}, {Vajab eluks vett, Elab maa peal})
9. ({Pingviin, Kotkas}, {Vajab eluks vett, Elab maa peal, Lind})
10. ({Kotkas}, {Vajab eluks vett, Elab maa peal, Lind, Lendab})
11. ({Pingviin, Inimene}, {Vajab eluks vett, Elab maa peal, Liigub kahel jalal})
12. ({Vesiroos}, {Vajab eluks vett, Elab vees, Taim})
13. ({Tamm}, {Vajab eluks vett, Elab maa peal, Taim})
14. ({Pingviin}, {Vajab eluks vett, Elab maa peal, Liigub kahel jalal, Lind})
15. ({}, {Vajab eluks vett, Taim, Elab maa peal, Elab vees, Imetaja, Lind, Lendab, Liigub kahel jalal})

Kontseptid on omavahel järjestatud moodustades ülem- ja alamkontsepti suhte. Alamkontsept sisaldab endas kõiki ülemkontsepti atribuute ning ülemkontsept sisaldab endas kõiki alamkontsepti objekte. Alamkontsept on oma ülemkontseptist spetsiifilisem, sest alamkontseptil on lisaks kõigile ülemkontsepti atribuutidele ka atribuute, mida ülemkontseptil ei ole. Sel moel moodustavad kontseptid omavahelise hierarhia. Näiteks ({Koer, Karu, Inimene}, {Vajab eluks vett, Elab maa peal, Imetaja}) on ülemkontsept kontseptile ({Inimene}, {Vajab eluks vett, Elab maa peal, Imetaja, **Liigub kahel jalal**}).

2.3 Kontseptivõre

Kontekstis leiduvate kontseptide visuaalseks esitamiseks kasutatakse kontseptivõresid. Lisaks kontseptide endi visualiseerimisele kujutab võre konteksti struktureeritud kujul näidates kontseptide vahelist hierarhiat ning struktuuri. Joonis 1-l on kujutatud Tabel 1-s esitatud konteksti kontseptivõre. Kontseptivõre iga tipp esitab ühte kontsepti ning võre kaared kujutavad kontseptide vahelisi seoseid. Tipu kohal hallil taustal on kujutatud kontsepti atribuudid ning tipu all valgel taustal on kujutatud kontsepti objektid. Kontseptivõre loetavamaks kujutamiseks kuvatakse igat konteksti objekti ja atribuuti ainult üks kord. Leidmaks kontsepti kogu ekstensiooni tuleb liikuda võres sellest kontseptist allapoole ning kõik vastutulevad objektid kuuluvad sellesse kontsepti. Analoogselt, leidmaks kontsepti kõiki atribuute tuleb liikuda võres kontseptist ülespoole ning kõik vastutulevad atribuudid kuuluvad sellesse kontsepti. Kontseptivõre kaarte ristumine väljaspool tippe ei oma mingit tähendust.



Joonis 1. Kontseptivõre.

Liikudes kontseptist mööda kaart alla jõutakse selle kontsepti alamkontseptini ning liikudes mööda kaart üles jõutakse ülemkontseptini. Alam-ülemkontsept suhe on oma olemuselt transitiivne, mis tähendab, et kõik vaadeldavast kontseptist allapoole jäävad

kontseptid on tema alamkontseptid ning kõik ülespoole jäävad kontseptid on tema ülemkontseptid [1].

Võre esimese kontsepti ekstensioon sisaldab kõiki konteksti objekte ning võre viimase kontsepti intentsioon sisaldab kõiki konteksti atribuute. Antud konteksti korral on kõikidel objektidel üks ühine atribuut ning seega kuulub see atribuut ka esimesse kontsepti. Juhul, kui kontekstis on atribuudid, mis teineteist välistavad, siis viimase kontsepti ekstensioon on alati tühi. Tabel 1-s esitatud konteksti puhul on teineteist välistavateks atribuutideks näiteks ’elab maa peal’ ning ’elab vees’, sest ühelgi objektil ei ole mõlemat atribuuti.

Kontseptiahel on kõigi kontseptide alamhulk, mis on lineaarselt järjestatud ning iga kontsepti paari vahel on ülem-alamkontsepti suhe [4]. Kontseptivõres moodustavad kontseptiahela kõik kontseptid, mis jäävad teele liikudes võre esimesest kontseptist võre viimase kontseptini. Igal kontseptiahela kontseptil on ainult üks vahetu ülem- ja alamkontsept, v.a kontseptiahela esimesel ja viimasel kontseptil.

Kõik selles magistritöös kujutatud kontseptivõred on loodud kasutades ConExp tööriista[1]. Lisaks kontseptivõrede visualiseerimisele võimaldab see leida konteksti kontseptide vahelisi implikatsioone.

2.4 Kontsepti stabiilsus

Kontsepti stabiilsus kirjeldab kuivõrd palju sõltub kontsepti intentsioon kontsepti ekstensioonist ehk kui eemaldada kontseptist mingi objekt, kas siis kontsepti intentsioon jääb muutumatuks või mitte. Kui kontseptilt eemaldada mõni objekt, siis kõrgema stabiilsusega kontsepti puhul intentsiooni muutumine on vähem tõenäoline kui madala stabiilsusega kontsepti puhul [5]. Stabiilsuse arvutamise valem on järgnev:

$$\sigma(A, B) = \frac{|\{C \subseteq A \mid C' = B\}|}{2^{|A|}}$$

[1] http://conexp.sourceforge.net

2.5 FCA kasutusalad

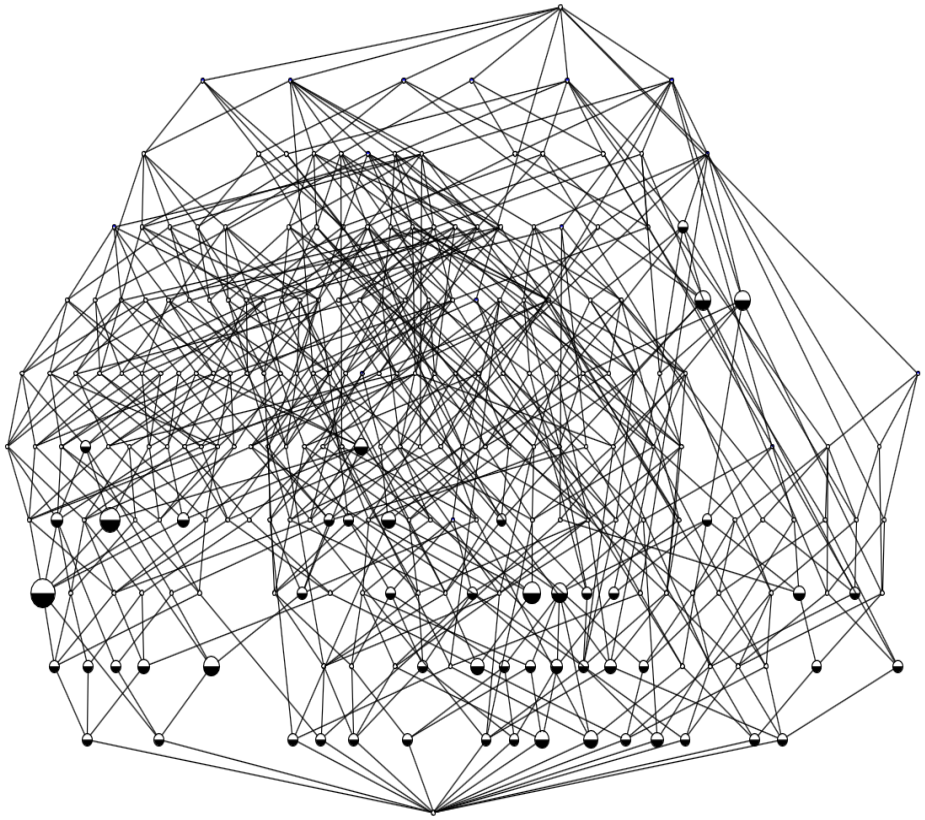
Kuigi formaalne kontseptianalüüs on võrdlemisi uudne valdkond, siis sellegipoolest on seda rakendatud mitmes erinevas valdkonnas. Lisaks andmete visualiseerimisele ning andmete vaheliste seoste otsimisele kontseptivõrede abil on FCA baasil loodud ka mitmeid erinevaid rakendusi informatsiooni esitamiseks, analüüsimiseks ning otsimiseks. Üheks kasutusalaks on dokumentide struktureerimine ning otsing märksõnade (dokumentide atribuutide) järgi, näiteks on FCA-d rakendatud juriidilises valdkonnas dokumentide otsimiseks [6]. Samuti on FCA abil loodud soovitussüsteeme, kus kasutatakse kontseptivõre struktuuri soovituste leidmiseks [7]. Sarnaselt on kontseptianalüüsi kasutatud versioonihalduses hoiustatud programmikoodi ja selles tehtavate muudatuste visualiseerimiseks ning selle põhjal info leidmiseks ning järelduste tegemiseks. Selle lahenduse abil on võimalik paremini mõista projekti struktuuri, näiteks saab lihtsa vaevaga teada, kes arendajatest tegeleb mingi allsüsteemiga või millised arendajad teevad omavahel tihedalt koostööd [8]. Samuti on FCA-d ning kontseptivõrede struktuuri rakendatud kontseptuaalsel klasterdamisel, kus kontseptivõres on objektid esitatud hierarhiana ning sarnased objektid asuvad võres lähestikku [9].

3 Seotud tööd

Selles peatükis tutvustatakse lähemalt magistritöös lahendatavat probleemi ning antakse lühike ülevaade erinevatest olemasolevatest meetoditest, mis üritavad seda probleemi lahendada.

3.1 Probleem

Kontseptivõre esitab kontsepte küll intuitiivsel ning struktureeritud kujul, kuid suure kontseptide arvu korral muutub see väga raskesti loetavaks. Näiteks Lisa 2-s toodud loomaaia kontekstis [10] on 100 objekti ning 15 atribuuti, kuid selle põhjal loodud kontseptivõre (Joonis 2) koosneb juba 238-st kontseptist ning on ka ilma objektide ja atribuutide kuvamiseta väga raskesti loetav.



Joonis 2. Kontseptivõre, milles on 238 kontsepti.

Käesoleva magistritöö üheks eesmärgiks on esitada kontekst kompaktsemal kujul ning muuta kontseptivõrede esitamine loetavamaks ja hoomatavamaks. Selle jaoks saab kasutada kolme erinevat lähenemist:

1. Eemaldada üleliigne informatsioon selliselt, et muuta kontekst väikeseks eemaldades võimalikult palju objekte ja atribuute, kuid seeläbi võre struktuuri muutmata.

2. Lihtsustada kontseptivõret muutes see abstraktsemaks ehk näidata ainult võre kõige tähtsamaid aspekte.

3. Valida neid kontsepte, objekte või atribuute, mis vastavad mingile relevantsuse kriteeriumile [11].

Järgnevas peatükis kirjeldatakse selle probleemi lahendamiseks erinevaid olemasolevaid meetodeid, mis kasutavad eelpool mainitud lähenemisi.

3.2 Olemasolevad lahendused

Üheks võimaluseks on rakendada kontseptide filtreerimist, kus leitakse objektid, millel on suur hulk ühiseid atribuute teiste objektidega. Selliseid objekte nimetatakse regulaarseteks objektideks ning neid objekte, mis ei ole regulaarsed, nimetatakse marginaalseteks. Kontseptivõre lihtsustamiseks filtreeritakse kontekstist välja kõik marginaalsed objektid. Allesjäänud objektide seast leitakse jälle kõik marginaalsed objektid ning nad eemaldatakse. Seda protsessi jätkatakse kuni andmehulgas ei ole enam ühtegi marginaalset objekti peale mida luuakse kontseptivõre vähendatud konteksti põhjal [12].

Kontseptivõre kompaktsemaks esitamiseks saab kasutada ka kontseptide eemaldamist sarnasuse põhjal. Sul juhul kõik kontekstis leiduvad objektid jagatakse sarnasuse järgi klastritesse ning iga klastri seast valitakse üks objekt, mis seda klastrit kõige paremini iseloomustab ning ülejäänud klastri objektid eemaldatakse. Objektide sarnasust hinnatakse nende atribuutide kaalude summa järgi. Iga kontekstis oleva atribuudi kaal, mis on vahemikus 0 kuni 1, iseloomustab selle atribuudi relevantsust. Lisaks on võimalik valida, mis võib olla klastri maksimaalne suurus. Meetod eeldab, et selle kasutaja tunneb oma algandmeid piisavalt hästi ja suudab anda atribuutidele adekvaatseid kaale, sest atribuudi kaalu automaatselt ei leita, vaid see tuleb määrata käsitsi [13]. On olemas ka sarnane lahendus, kus atribuutidele ei pea kaalusid andma, vaid peab valima, mitu klastrit soovitakse leida ning selle põhjal luuakse uus kontseptivõre [14]. On pakutud ka mõnevõrra sarnast objekti kaalusid kasutavat

lahendust, kus igale kontseptile leitakse kaal ning kontseptivõres kuvatakse vaid neid kontsepte, mille kaal on piisavalt kõrge. Igal atribuudil on olemas mingi kaal ning selle kaudu leitakse ka kontsepti kaal summeerides kõigi kontsepti intentsioonis olevate atribuutide kaalud. Nagu ka eelneva lahenduse puhul peab siingi atribuutide kaalud määrama käsitsi [15].

Samuti on võimalik esitada ka ainult väike osa kogu kontseptivõrest kasutades jäämäe tüüpi kontseptivõresid. Jäämäe tüüpi kontseptivõre koosneb algse kontseptivõre ülemistest kontseptidest, milleks on kõik sagedased kontseptid. Kontsepti nimetatakse sagedaseks, kui tema intentsioon on sagedane ehk see intentsioon esineb paljudel kontseptidel. Selle lähenemise puhul defineeritakse mingi piirarv, millest alates loetakse kontsepti sagedaseks. Sageduse piirmäära vähendades muutub kontseptivõre 'jäämäe' osa suuremaks. Kontsepti intentsiooni sageduse leidmiseks jagatakse kontsepti objektide arv kogu konteksti objektide arvuga. Näiteks teadustöös, mis seda meetodit tutvustab, kasutati näitena konteksti, mis sisaldab 32 086 kontsepti. Selle konteksti puhul koosneb võre jäämäe osa sagedusel 85% 7-st kontseptist, sagedusel 70% 12-st kontseptist ning sagedusel 55% 32-st kontseptist [16].

Üheks triviaalseks lahenduseks on vähendada konteksti suurust. Vähendades konteksti atribuute saab luua alamkonteksti, mis sisaldab ainult neid atribuute, mille vahelisi seoseid soovitakse uurida. Sarnaselt atribuutide eemaldamisele saab luua alamkonteksti, mis sisaldab ainult valituid objekte. Objektide valik toimub atribuutide järgi: valitakse ainult need objektid, millel on olemas mingid atribuudid. Eemaldamised toimuvad vastavalt sellele, milliste objektide ning atribuutide vahelisi seoseid andmetega töötaja näha soovib. Samuti on võimalik vähendada konteksti müra valides ainult neid kontsepte, millel on olemas minimaalne arv objekte ja atribuute. Müraks loetakse väikeseid kontsepte. Konteksti müra vähendamine on oma olemuselt sarnane jäämäe tüüpi kontseptivõredega [17].

Mõnevõrra sarnaselt jäämäe võre leidmise lähenemisele on pakutud ka meetodit, kus meetodi kasutaja saab ise seada kitsendusi, mille abil saab määrata, milliseid kontsepte tuleb kontseptivõres kuvada. Selle lähenemise puhul peab kasutajal olema mingi teadmine uuritava andmestiku struktuurist. Kitsenduseks võib olla näiteks mõne objekti või atribuudi olemasolu või puudumine [18]. Samuti on võimalik näidata kontseptivõres

ainult neid kontsepte, mille stabiilsus on piisavalt kõrge. Stabiilsus, mis peab kontseptil olema, et ta oleks kontseptivõres kuvatud, on määratud kasutaja poolt [19].

Võimaluseks on teisendada kontseptivõre puu kujule. Nimelt kontseptivõrest moodustatakse kontseptipuu, kus igal kontseptil (v.a esimene kontsept) on ainult üks vahetu ülemkontsept. Meetodi rakendamist alustatakse võre viimasest kontseptist ning liigutakse võres üles valides igale järgnevale kontseptile tema vanemkontsept. Rakendades seda meetodit kaob osa algses kontseptivõres olevast informatsioonist, sest eemaldatakse mõnede kontseptide vahelised kaared, kuid sellegipoolest on võre struktuur semantiliselt korrektne, sest ühtegi kontsepti, objekti ega atribuuti välja ei võeta. Informatsioonikadu saab vähendada, kui vanemkontsepti valida kindla kriteeriumi järgi. Kontsepti valimise kriteeriume on neli: suurim kontsepti stabiilsus, suurim kontseptide ühiste atribuutide hulk, tõenäosus, lühim tee võre esimese kontseptini [20].

Kontseptivõrede lihtsustamiseks on kasutatud ka graafiteooriat. Graafide puhul saab mingi kindla tippude hulga, mida nimetatakse mooduliks, asendada väiksema tippude hulgaga, mis esindab tervet moodulit. Väikeste muudatustega saab sama lähenemist kasutada ka kontseptivõrede puhul [21].

Lähtudes eelpool kirjeldatud meetoditest on näha, et kontseptivõre suuruse ja keerukuse vähendamiseks on üritatud leida hulganisti erinevaid lahendusi, kuid mitte ükski neist ei ole veel muutunud *de facto* standardiks, mis kõigile sobiks. Seega on selle töö eesmärk üpriski aktuaalne ning õigustatud, sest esitatakse veel üks lähenemine olemasoleva probleemi lahendamiseks.

4 Töös kasutatud tehnikad

Käesolevas peatükis vaadeldakse töös kasutatavaid tehnikaid andmete töötlemiseks ning tutvustatakse olemasolevat kontseptiahelate leidmise algoritmi, mida hakatakse selles töös edasi arendama. Samuti kirjeldatakse, kuidas hakatakse hindama leitud kontseptiahelate headust.

4.1 Klasterdamine

Magistritöös loodavas algoritmis kasutatakse mitme kontseptiahela genereerimisel ühe sammuna ka andmete klasterdamist. Klasterdamine on objektide grupeerimine selliselt, et samas grupis olevad objektid on teineteise suhtes sarnased ning teistes gruppides olevate objektide suhtes vähem sarnased. Kuna antud töös tegeletakse binaarse andmestikuga, siis tuleb valida klasterdamiseks algoritm, mis suudab klasterdada binaarseid andmeid. Klasterdamise võib jagada kaheks: hierarhiline klasterdamine ning mittehierahiline klasterdamine. Järgnevalt tutvustatakse mõningaid võimalikke algoritme, mida kaaluti töös kasutada. Töös kasutatud meetodit ning selle valimise põhjuseid tutvustatakse töö praktilises osas.

K-means on klasterdamise algoritm, mis on mõeldud töötamaks objektidega, mille kõik atribuudid on arvulised ning objektide vahelised kaugused ehk objektide erinevus arvutatakse eukleidilise kauguse abil. Algoritm võtab sisendiks soovitud klastrite arvu *k* ning loob *k* punkti, mis tähistavad iga klastri keskpunkti. Seejärel pannakse kõik objektid lähimasse klastritesse. Järgmisena arvutatakse uuesti iga klastri keskpunkt objekti atribuutide keskmise väärtuse järgi ning kui mõne objekti jaoks ei ole tema olemasolev klastri keskpunkt lähim, siis viiakse see objekt talle lähimasse klastrisse. Keskpunktide arvutamine ning objektide liigutamine kestab niikaua kuni ühegi objekti asukoht ei muutu [22]. Kuna *k-means* töö põhineb objekti atribuutide keskmise väärtuse järgi, siis see ei sobi binaarsete andmetega töötamiseks, kuid magistritöö kasutatavad andmed on binaarsel kujul. Binaarsete andmete klasterdamiseks on loodud *k-modes* algoritm, mis on sarnane *k-means* algoritmile. *K-modes* erineb sellepoolest, et see ei kasuta klasterdamisel objektide atribuutide keskmist väärtust, vaid atribuutide moodi [23] ning eukleidilise kauguse asemel kasutatakse objektide vahelist Hammingu kaugust [24], mis näitab, mitu atribuuti on kahel objektil erineva väärtusega. Nii *k-modes*'i [25]

kui ka *k-means*'i [26] keerukuseks on $O(mnki)$, kus m on objektide arv, n on atribuutide arv, k on otsitavate klastrite arv ning i on klastrite leidmisel tehtud iteratsioonide arv.

Hierarhiline klasterdamine on sobilikum variant juhul, kui andmestikul on olemas mingisugune süstemaatiline struktuur. Samuti sobib see binaarsete andmetega töötamiseks [27]. Eelnevatest peatükkidest selgus, et formaalses kontekstis olevad andmed omavad struktuuri, seega on hierarhiline klasterdamine antud töö puhul sobilik lähenemine. Hierarhilise klasterdamise puhul moodustatakse klastritest hierarhia, kus igal tasemel olev klaster on loodud alumisel tasemel olevate klastrite ühendamisel. Madalaimal tasemel koosneb iga klaster ühest objektist. Hierarhilisel klasterdamisel on kaks erinevat lähenemist:

- Iga objekt moodustab ühe klastri ning seejärel hakatakse klastreid ühendama kuni on leitud vajalik arv klastreid.
- Kõik objektid moodustavad ühe klastri ning seejärel hakatakse seda klastrit jagama väiksemateks klastriteks [22].

Hierarhilise klasterdamise keerukus on $O(n^2)$ [28].

DBSCAN (*Density based spatial clustering of applications with noise*) jaotab andmed klastritesse tiheduse põhjal. Esmalt leitakse üks põhiobjekt, millest saab klastri keskpunkt ning seejärel leitakse kõik objektid, mis on keskpunktist kindlal kaugusel ning mis moodustavad klastri. Klastrite leidmiseks ei ole vaja teada klastrite arvu, vaid on vaja kahte parameetrit: klastris olevate minimaalne objektide arv ning maksimaalne kaugus, mis võib olla klastri keskpunkti ning iga klastris oleva objekti vahel. Objektid, mis ei ole piisavalt lähedal ühelegi klastri keskpunktile liigitatakse müraks. Algoritmi keerukus on $O(n * logn)$ [29].

4.2 Monotoonsed süsteemid

Monotoonne süsteem koosneb lõplikust objektide hulgast ja objekti kaalu arvutamise monotoonsest funktsioonist, mille abil arvutatakse objekti tähtsus vaadeldavas süsteemis. Monotoonseid süsteeme kasutatakse objektide järjestamiseks, mille kaudu avalduvad objektide mustrid [4]. Antud töös on süsteemiks formaalne kontekst.

Objektide kaalud vähenevad monotoonselt, kui mõni objekt süsteemist eemaldatakse. Järgnevalt tutvustatakse konformismiskaalat, mida kasutatakse objektide kaalude leidmiseks ning miinustehnikat, mis kasutab konformismiskaalat objektide informatiivsemaks järjestamiseks.

4.2.1 Konformismiskaala

Esmalt arvutatakse iga atribuudi esinemissagedus kontekstis. Selle järgi arvutatakse objekti kaal liites kokku objektil olevate kaalude esinemissagedused [30]. Rakendades Tabel 1-s esitatud kontekstile konformismiskaalat saadakse kaalud, mis on kirjeldatud Tabel 2-s. Praktilises töö osas objektid sorteeritakse kaalu järgi kahanevas järjestuses, kuid Tabel 2-s objektide järjekorda muudetud ei ole. Konformismiskaala keerukuseks on $O(n)$, kus n on objektide arv.

Tabel 2. Konformismiskaala näide.

	Vajab eluks vett	Taim	Elab maa peal	Elab vees	Imetaja	Lind	Lendab	Liigub kahel jalal	*Objekti kaal*
Inimene	X		X		X			X	*21*
Pingviin	X		X			X		X	*19*
Kotkas	X		X			X	X		*18*
Karu	X		X		X				*19*
Koer	X		X		X				*19*
Tamm	X	X	X						*17*
Delfiin	X			X	X				*16*
Vesiroos	X	X		X					*14*
Ahven	X			X					*12*
Atribuudi kaal	*9*	*2*	*6*	*3*	*4*	*2*	*1*	*2*	

4.2.2 Miinustehnika

Miinustehnika on Eesti teadlaste poolt välja töötatud lähenemine andmetabelite ridade informatiivsemaks järjestamiseks. Miinustehnika abil järjestatakse andmetabeli objektid nende väärtuslikkuse järgi vaadeldavas süsteemis. Andmetabeli järjestamiseks kasutatakse konformismiskaalat ning protseduur on järgnev:

1. Leitakse olemasolevate objektide kaalud konformismiskaala abil.

2. Leitakse objekt, millel on väikseim kaal ning see objekt eemaldatakse jättes meelde, mitmendana see objekt eemaldati.

3. Niikaua, kuni tabelist ei ole kõik objektid eemaldatud, rakendatakse nende peal samme 1 kuni 2.

4. Andmetabel järjestatakse vastupidises objektide eemaldamise järjekorras ehk tabeli ülemises osas on tähtsamad objektid, mis olid eemaldatud viimasena [30].

Miinustehnika rakendamise keerukus on $O(n^2)$, kus n on objektide arv, sest iga objekti kohta tuleb konformismiskaala uuesti välja arvutada. Tabel 3 kujutab Tabel 1-s esitatud konteksti, millele on rakendatud miinustehnikat ning nagu näha, siis tabeli ülemises osas on objektid, millel on populaarsemad atribuudid ning sarnased objektid püsivad tabelis kõrvuti.

Tabel 3. Miinustehnika näide.

	Vajab eluks vett	Taim	Elab maa peal	Elab vees	Imetaja	Lind	Lendab	Liigub kahel jalal
Inimene	X		X		X			X
Pingviin	X		X			X		X
Kotkas	X		X			X	X	
Karu	X		X		X			
Koer	X		X		X			
Tamm	X	X	X					
Delfiin	X			X	X			
Vesiroos	X	X		X				
Ahven	X			X				

4.3 Kontseptiahelate leidmine

Käesoleva magistritöö eesmärgiks on täiendada Ants Torimi poolt loodud algoritmi, mis leiab kontseptivõres kontseptiahelaid ilma tervet kontseptivõret loomata ja kõiki kontsepte genereerimata. Kuna algoritm ei ole teadusartiklina ega ka muul kujul

avaldatud, siis järgnevalt kirjeldatakse algoritmi tööpõhimõtet ning esitatakse algoritmi pseudokood.

Algoritm võtab sisendiks konteksti, kus atribuudi olemasolu objektil on tähistatud numbriga 1 ning atribuudi puudumine numbriga 0. Seejärel rakendatakse kontekstile miinustehnikat ning saadakse optimaalne objektide järjekord. Peale seda hakatakse genereerima kontseptiahelat ehk leitakse hulk kontsepte, mis viivad võre esimesest kontseptist võre viimase kontseptini. Kontseptiahela leidmise algoritmi pseudokood on kujutatud Joonis 3-l.

Ahela genereerimine algab võre viimasest kontseptist. Selleks defineeritakse kaks hulka, mida kasutatakse kogu algoritmi töö käigus: üks objektide jaoks ning teine atribuutide jaoks. Need hulgad tähistavad hetkel loodava kontsepti ekstensiooni ja intentsiooni. Ekstensioon on algselt tühi ning intentsioon sisaldab algselt kõiki kontekstis leiduvaid atribuute. Seejärel hakatakse looma kontsepte itereerides üle kõigi objektide, mis on miinustehnika abil järjestatud. Kui itereeritava objekti kõik atribuudid on atribuutide hulgas olemas, siis lisatakse see objekt objektide hulka (rida 13). Juhul, kui itereeritaval objektil ei ole kõiki atribuutide hulgas leiduvaid atribuute, siis see tähendab, et see objekt ei kuulu hetkel loodavasse kontsepti, vaid ta kuulub võres ülemisse (eelmisesse) kontsepti. Sel juhul on leitud üks kontsept, mis lisatakse kontseptiahelasse. Peale seda hakatakse looma uut kontsepti ning muudetakse objektide ja atribuutide hulka: objektide hulka lisatakse praegune objekt (rida 10) ning atribuutide hulka jäetakse ainult need atribuudid, mis on ühised just loodud kontseptil ning praegusel objektil (rida 11). Protsessi jätkatakse kuni kõik objektid on läbi käidud. Protsessi lõppedes tagastatakse kontseptide hulk ehk kontseptiahel. Tasub märkida, et käesolev algoritm ei tagasta kontseptivõre esimest ega viimast kontsepti, kui nendes kontseptides on tühi intentsioon (esimene kontsept) või on tühi ekstensioon (viimane kontsept). Sellisel moel kontseptiahelat luues on väga tähtis roll konteksti objektide hulga järjestusel.

Kui seda algoritmi rakendada kaks korda: esimesel korral on objektid järjestatud miinustehnika abil ning teisel korral on objektid järjestatud suvaliselt, siis ei pruugi algoritm sama kontseptiahelat mõlemal korral leida. Miinustehnika tagab, et kontseptiahelas on võimalikult palju tähtsaid objekte ehk objekte, millel on kontekstis suurim kaal.

```
01: def find_chain(objects, all_attributes):
02:     concepts = []
03:     extent = set()
04:     intent = set(all_attributes)
05:     for obj in objects:
06:         obj_intent = set(obj.attributes)
07:         if len(intent - obj_intent) > 0:
08:             if len(extent) > 0:
09:                 concepts.append((extent, intent))
10:             extent = extent | {obj}
11:             intent = intent & obj_intent
12:         else:
13:             extent.add(obj)
14:     if len(intent) > 0:
15:         concepts.append((extent, intent))
16:     return concepts
```

Joonis 3. Kontseptiahela leidmise algoritm.

4.3.1 Kontseptiahela leidmise näide

Järgnevalt rakendatakse kontseptiahela leidmise algoritmi Tabel 1-s kirjeldatud loomade konteksti peal. Konteksti objektid on miinustehnika abil järjestatud eelmises alapeatükis Tabel 3-s. Vaadeldava konteksti puhul on 9 objekti seega algoritm teeb 9 iteratsiooni ning järgnevalt esitatakse algoritmi iga iteratsioon eraldi punktina.

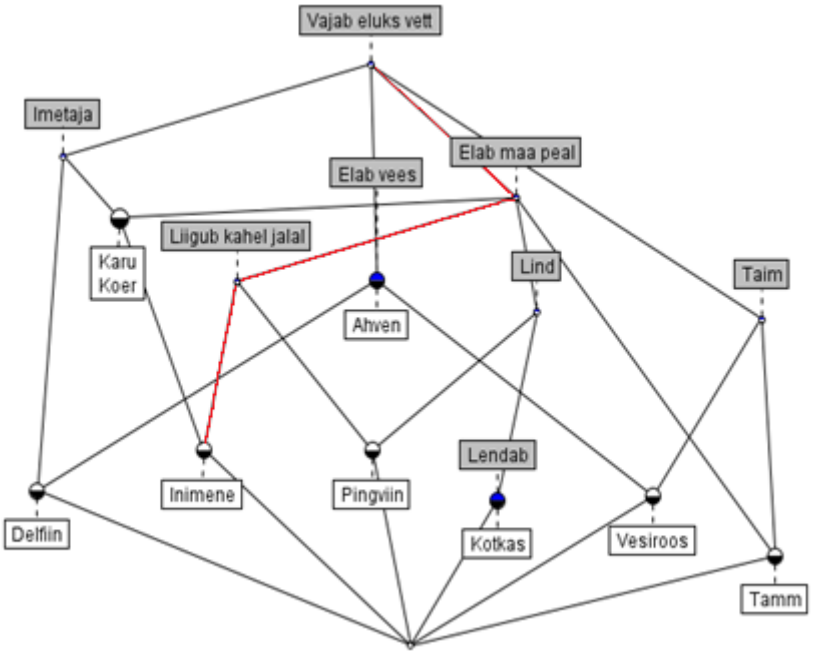
1. Hetkel on ekstensioon on tühi ning intentsioonis on kõik atribuudid. Võetakse ette esimene objekt, milleks on *inimene* ning leitakse kõik selle objekti atrbuudid. Kuna hetkel on objektil atribuute, mida ei ole intentsioonis, siis järelikult on leitud uus kontsept. Uue kontsepti esimeseks objektiks on *inimene*. Muutuja *extent* sisaldab objekti *inimene* ning muutuja *intent* sisaldab kõiki selle objekti atribuute.
2. Järgnevalt vaadeldakse objekti *pingviin*. Ka sellel objektil ei ole kõiki atribuute, mis on praeguses intentsioonis, seega kuulub see objekt uude kontsepti. Samuti tähendab see, et eelmises iteratsioonis loodud kontsept sisaldab ainult ühte objekti ning sellest saab kontseptiahela viimane kontsept. Selles iteratsioonis luuakse uus kontsept, milles on kaks objekti: *inimene* ning *pingviin*. Kontsepti intentsiooniks on kõik atribuudid, mis on neile kahele objektile ühised, milleks on *vajab eluks vett, elab maa peal, liigub kahel jalal*.

3. Järgnevalt vaadeldakse objekti *kotkas*. Ka sellel objektil ei ole kõiki atribuute, mis on praeguses intentsioonis, seega kuulub see objekt uude kontsepti. Eelmises punktis alguse saanud kontsept enam uusi objekte juurde ei saa ning sellest saab kontseptiahela eelviimane kontsept. Selles iteratsioonis luuakse uus kontsept, milles on kolm objekti: eelmise kontsepti objektid ning *kotkas*. Intentsiooniks on kolme objekti ühised atribuudid: *vajab eluks vett, elab maa peal*.
4. Järgnevalt vaadeldakse objekti *karu*. Sellel objektil on olemas kõik atribuudid, mis on olemas eelmises punktis loodud kontseptis. Seetõttu uut kontsepti ei looda, vaid lisatakse see objekt eelmise kontsepti ekstensiooni.
5. Järgnevalt vaadeldakse objekti *koer*. Selle objektiga on olukord sama, mis eelmisegagi: lisatakse ekstensiooni.
6. Järgnevalt vaadeldakse objekti *tamm*. Selle objektiga on olukord sama, mis eelmisegagi: lisatakse ekstensiooni.
7. Järgnevalt vaadeldakse objekti *delfiin.* Sellel objektil ei ole kõiki atribuute, mis on kontseptis olemas, nimelt ei ole objektil *delfiin* atribuuti *elab maa peal*. Seetõttu eelmistes punktides suurendatud kontsept on valmis ning *delfiin* moodustab uue kontsepti. Uus kontsept sisaldab kõiki senimaani nähtud objekte ning atribuudiks on ainult *vajab eluks vett*.
8. Järgnevalt vaadeldakse objekti *vesiroos*. Sellel objektil on olemas kõik kontsepti intentsiooni atribuudid, seega lisatakse see kontsepti ekstensiooni.
9. Järgnevalt vaadeldakse viimast objekti *ahven*. Ka see objekt lisatakse ekstensiooni. Kuna rohkem objekte ei ole, siis on leitud ka viimane kontsept. Kontseptiahel moodustubki kolmest leitud kontseptist.

Kokkuvõttes saadakse kontseptiahelaks järgnev kolmest kontseptist koosnev kontseptiahel (kontseptid on järjestatud alates kõige ülemisest kontseptist kuni alumise kontseptini):

({delfiin, koer, karu, ahven, inimene, pingviin, kotkas, vesiroos, tamm}, {vajab eluks vett}) →
({koer, karu, inimene, pingviin, tamm, kotkas}, {vajab eluks vett, elab maa peal}) →
({pingviin, inimene}, {vajab eluks vett, elab maa peal, liigub kahel jalal}) →
({inimene}, {vajab eluks vett, imetaja, elab maa peal, liigub kahel jalal})

Joonis 4-l on kujutatud leitud kontseptiahel punase joonena. Kuna antud konteksti puhul esimese kontsepti intentsioon ei ole tühi, siis sisaldub see ka leitud kontseptiahelas.



Joonis 4. Kontseptiahel kontseptivõres.

4.4 Kontseptiahela katvus

Kuna kontseptiahela leidmise eesmärgiks on esitada uuritav kontekst kompaktsel kujul, siis peab kasutama mingisugust meetodit, mis näitaks, kuivõrd suurt osa kontekstist saadud kontseptiahel katab. Selle jaoks kasutatakse töös kontseptide poolt kaetud objektide ja atribuutide vahelist relatsioonide arvu suhet kõigi kontekstis olevate relatsioonidega [4].

Näiteks Tabel 1-s kuvatud konteksti puhul on kokku 29 objekti ja atribuudi vahelist relatsiooni. Eelnevalt leitud kontseptiahelas on kokku 4 kontsepti, mis katavad ära 18 relatsiooni ning seega on kontseptiahela poolt kaetud 62% kogu kontekstist. Tabel 4-s on näidatud kaetud relatsioonid punase ristiga. Kontseptiahela katvuse abil saab hinnata, kuidas mitme kontseptiahela leidmisel muutub konteksti katvus ning alates mitmendast kontseptiahelast uute kontseptiahelate leidmine enam katvust märgatavalt ei tõsta ning kasu ei too. Samuti saab selle abil hinnata, kui suur osa algselt kontekstis esitatud informatsioonist läheb meetodi rakendamisel kaduma.

Tabel 4. Kontseptiahela katvus

	Vajab eluks vett	Taim	Elab maa peal	Elab vees	Imetaja	Lind	Lendab	Liigub kahel jalal
Koer	X		X		X			
Karu	X		X		X			
Tamm	X	X	X					
Pingviin	X		X			X		X
Ahven	X			X				
Vesiroos	X	X		X				
Delfiin	X			X	X			
Kotkas	X		X			X	X	
Inimene	X		X		X			X

5 Mitme kontseptiahela leidmine

Praktilise töö eesmärgiks on arendada edasi kontseptiahela leidmise algoritmi. Edasiarenduse eesmärkideks on lisada funktsionaalsus, mis võimaldab leida mitut kontseptiahelat ning samuti olemasolevat algoritmi optimeerida, et kontseptiahela leidmine võtaks vähem aega. Järgnevates peatükkides kirjeldatakse algoritmi üldist põhiideed ning seejärel tutvustatakse igat algoritmi osa põhjalikult eraldi. Samuti tutvustatakse algoritmi loomiseks kasutatud tehnoloogiaid.

5.1 Kasutatud tehnoloogiad

Algoritm on realiseeritud kasutades Python 3 programmeerimiskeelt, sest sellega saab autori arvates kiiresti valmis teha töötava prototüübi, samuti oli selle abil juba loodud ühe kontseptiahela leidmise prototüüp ning selle jaoks on saadaval erinevaid populaarseid andmeteaduse jaoks mõeldud teeke, mis teevad näiteks maatriksitega töötamise arendaja jaoks lihtsaks ning pakuvad erinevaid klasterdamise algoritme. Kuigi Python on võrdlemisi aeglane võrreldes näiteks Javaga ning C-ga, siis Pythoni eeliseks on asjaolu, et koodi kirjutamisele saab kulutada vähem aega ning keskenduda rohkem käesoleva probleemi lahendamisele.

Töös on kasutatud järgnevaid Python'i teeke: *numpy*[1], mille abil saab teha arvutusi maatriksitega, *pandas*[2], mis pakub andmestruktuure andmetabelitega töötamiseks, *sklearn*[3], mille abil teostatakse hierarhilist klasterdamist ning *k-modes*[4], mille abil teostatakse *k-modes* klasterdamist. Leitud kontseptiahelate visualiseerimiseks kasutatakse eelnevalt mainitud ConExp tarkvara.

[1] http://www.numpy.org/

[2] http://pandas.pydata.org/

[3] http://scikit-learn.org

[4] https://github.com/nicodv/kmodes

5.2 Algoritmi seletus

Võrreldes ühe kontseptiahela leidmisega on mitme kontseptiahela leidmiseks vaja teha veidi rohkem eeltööd ning samuti tuleb muuta olemasolevat algoritmi. Kogu algoritmi kood on toodud Lisa 1-s.

Mitme kontseptiahela leidmise põhiidee seisneb selles, et esmalt leitakse kontseptivõres iga leitava kontseptiahela jaoks algkontseptid. Algkontseptideks nimetatakse neid sellepärast, et tegemist on kontseptidega alates millest toimub kontseptiahela leidmine. Erinevalt algsest algoritmist, kus kontseptiahela genereerimine algab kontseptivõre viimasest kontseptist ning lõpeb kontseptivõre esimese kontsepti leidmisega, siis mitme kontseptiahela puhul see nii ei ole. Algkontsept ei pruugi olla võre viimane kontsept, vaid võib asuda ka kusagil kontseptivõre keskel. Selleks, et olemasolevat algoritmi saaks võimalikult palju taaskasutada, leitakse kontseptiahel kahes osas. Nimelt leitakse esmalt kontseptiahel, mis viib algkontseptist kontseptivõre esimese kontseptini ning seejärel leitakse kontseptiahel, mis viib kontseptivõre viimasest kontseptist algkontseptini. Seejärel poolikud kontseptiahelad ühendatakse ning moodustub täielik kontseptiahel, mis viib kontseptivõre esimesest kontseptist viimase kontseptini.

Arvestades, et algkontseptid on kontseptiahela leidmise alustalaks, siis on vaja, et nad oleksid võimalikult erinevad. Erinevus on tarvilik, et algkontseptid asuksid kontseptivõres teineteisest võimalikult kaugel. Teineteisest kaugel asumine tähendab, et neil on vähe ühiseid atribuute. Sellisel juhul on ka kontseptiahelad võimalikult erinevad kattes ära võimalikult erinevad kontseptivõre osad ning erinevates kontseptiahelates samad kontseptid korduvad võimalikult vähe.

Algkontseptid leitakse klasterdamise teel. Esmalt klasterdatakse konteksti objektid ning igast klastrist võetakse üks objekt, millel on väikseim arv atribuute. Selline objekti valik tagab algkontsepti asumise võre ülemises osas. Leitud objekti põhjal luuakse kontsept, mille intentsiooniks on leitud objekti atribuudid ning ekstensiooniks on kõik konteksti objektid, millel on kõik leitud objekti atribuudid olemas. Klastrite arv sõltub sellest, kui palju kontseptiahelaid leida soovitakse, sest igast klastrist leitud algkontseptist leitakse üks kontseptiahel.

Kokkuvõtvalt võib kontseptiahelate leidmise algoritmi jagada järgmistesse sammudesse:

1. **Andmete lugemine**. Kontekst loetakse CSV failist.

2. **Andmete klasterdamine**. Kontekstis olevad objektid klasterdatakse, et leida võimalikult erinevad objektid, mille põhjal leitakse kontseptid. Klasterdamise vajadus seisneb selles, et nõnda leitud kontseptid on teineteisest võimalikult erinevad ning asuvad kontseptivõres teineteisest võimalikult kaugel. See tagab asjaolu, et leitavad kontseptiahelad on võimalikult erinevad.

3. **Algkontsepti loomine**. Eelnevas sammus leitud klastritest võetakse igast klastrist üks objekt, mille põhjal luuakse kontsept. Nõnda leitud kontseptide näol on tegemist algkontseptidega, mis asuvad kontseptivõres.

4. **Kontseptiahela leidmine**. Eelmises sammus leitud algkontseptidest hakatakse otsima kontseptiahelaid. Kuna algkontsept asub kontseptivõre keskel (ta ei ole ei esimene ega viimane kontsept), siis tuleb täieliku kontseptiahela leidmiseks leida 2 poolikut kontseptiahelat: üks, mis viib esimesest kontseptist algkontseptist ning teine, mis viib algkontseptist viimase kontseptini. Nõnda leitud poolikud kontseptiahelad ühendatakse ning moodustub terve kontseptiahel. Seda protsessi korratakse iga algkontsepti kohta eraldi.

5. **Kontseptiahelate katvuse arvutamine ning visualiseerimine**. Algoritmi hindamiseks arvutatakse kontseptiahelate katvus ning samuti on võimalik leitud kontseptiahelaid visualiseerida.

Alternatiivseks lahenduseks oleks võimalik genereerida terve kontseptivõre ning seejärel kasutades graafi otsingualgoritmi koos mõne heuristikaga, mis suudaks seatud tingimuste järgi parimaid kontseptiahelaid otsida. Antud lähenemise miinuseks on asjaolu, et selle eelduseks on kõikide kontseptide leidmine ning seejärel nendest kontseptivõre loomine, mis omakorda on keerukas ning ajakulukas protsess.

Järgnevalt tutvustatakse igat algoritmi sammu lähemalt.

5.2.1 Andmete lugemine

Algoritm võtab sisse andmed, mis on esitatud CSV formaadis, kus faili esimene rida koosneb atribuutidest ning esimene veerg koosneb objektidest. Objektide ning atribuutide vaheline seos võib olla märgitud X-ga või ka binaarsel kujul, kus 1 tähistab

atribuudi olemasolu ning 0 tähistab atribuudi olemasolu puudumist. Juhul, kui kontekst on kujutatud X-ga, siis teisendatakse see andmete lugemisel binaarsele kujule. Binaarne kuju on eelistatud, sest see võimaldab teha järgnevates peatükkides maatriksitega arvutusi. Tabel 1-s näidatud loomade kontekst näeb välja järgnevalt:

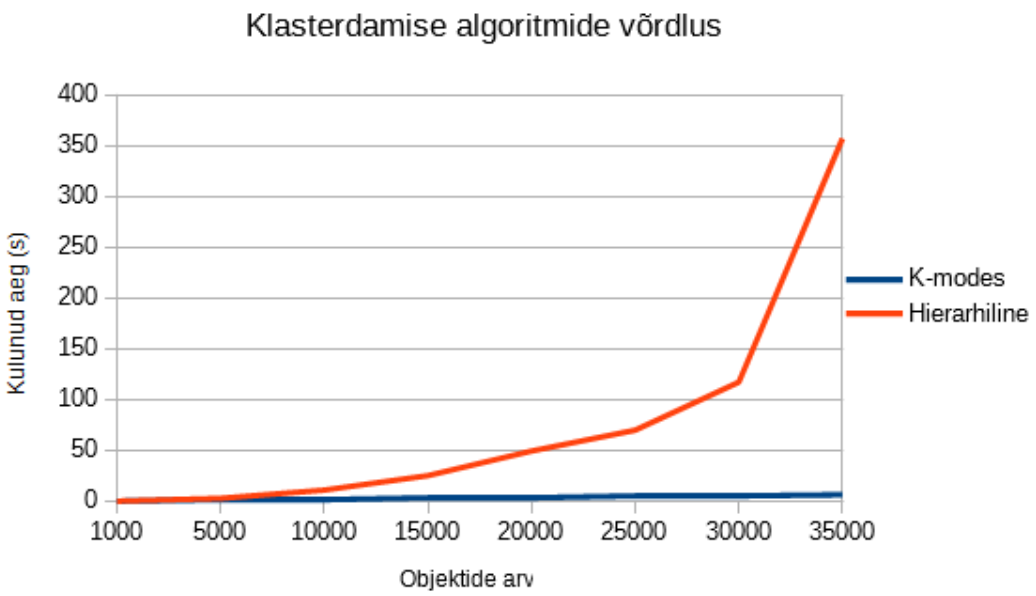
```
Vajab eluks vett, Taim, Elab maa peal, Elab vees, Imetaja, Lind, Lendab,
Liigub kahel jalal
Koer, 1, 0, 1, 0, 1, 0, 0, 0
Karu, 1, 0, 1, 0, 1, 0, 0, 0
Tamm, 1, 1, 1, 0, 0, 0, 0, 0
Pingviin, 1, 0, 1, 0, 0, 1, 0, 1
Ahven, 1, 0, 0, 1, 0, 0, 0, 0
Vesiroos, 1, 1, 0, 1, 0, 0, 0, 0
Delfiin, 1, 0, 0, 1, 1, 0, 0, 0
Kotkas, 1, 0, 1, 0, 0, 1, 1, 0
Inimene, 1, 0, 1, 0, 1, 0, 0, 1
```

Andmed loetakse sisse ning hoitakse *DataFrame* objektis, mis asub *pandas* teegis. Samuti andmed järjestatakse selliselt, et eespool olevad objektid oleksid suurema atribuutide arvuga kui tagapool asuvad objektid.

5.2.2 Andmete klasterdamine

Selleks, et leida võimalikult erinevad kontseptiahelad tuleb esmalt kontekstis olevad objektid klasterdada. Objektid klasterdatakse neil olemasolevate atribuutide järgi. Klasterdamise nõudeks on genereeritavate klastrite arvu valimise võimalus enne klasterdamist. Selline tingimus on seatud seetõttu, et iga klastri kohta genereeritakse üks kontseptiahel ning algoritmi kasutajal peab olema võimalus valida, mitu kontseptiahelat genereerida tuleb. Seetõttu sobivad klasterdamiseks ainult lähenemised, mis võimaldavad valida leitavate klastrite arvu.

Käesolevas töös otsustas autor klasterdamiseks kasutada *k-modes* algoritmi. DBSCAN ei sobi, sest selle lähenemise puhul ei ole võimalik valida klastrite arvu enne klasterdamist. Hierarhiline klasterdamise ei sobi, sest selle keerukus on suurem kui *k-modes* klasterdamise keerukus ja on seetõttu palju aeglasem. Kui kontekstis on juba mõnikümmend tuhat objekti, siis hierarhiline klasterdamine võtab märgatavalt rohkem aega kui *k-modes*. *K-modes*'i ning hierarhilise klasterdamise ajakulu võrdlus erineva objektide arvu korral on toodud alloleval graafikul. Võrdluses kasutatud kontekstides on 7 atribuuti, kontekstis on ühtede ja nullide arv võrdne ning andmed on genereeritud juhuslikult.



Joonis 5. Klasterdamise algoritmide võrdlus

K-modes'i algoritm võtab sisendiks andmete *numpy* massiivi, milleks on antud juhul konteksti relatsioonid ning leitavate klastrite arvu. Klasterdamise tulemuseks on numbrite massiiv, milles iga element tähistab klastrit, millesse samal indeksil asuv objekt kuulub. Leitavate klastrite arv peab olema vähemalt 1 ning ei tohi olla suurem kui on klasterdatavate objektide arv.

```
def cluster_data(self, data: np.ndarray, num_of_clusters: int) -> np.ndarray:
    import numpy as np
    from kmodes.kmodes import KModes
    clustering = KModes(n_clusters=num_of_clusters).fit(data)
    return clustering.labels_
```

Näiteks klasterdades Tabel 1-s kirjeldatud konteksti kolme klastrisse on tulemuseks järgnev massiiv:

```
[0 0 0 2 1 1 1 2 0]
```

Tulemusest selgub, et esimene objekt kuulub klastrisse numbriga 0 ning neljas objekt kuulub klastrisse numbriga 2. Ehk kolm leitud klastrit on:

1. Inimene, koer, karu, tamm
2. Pingviin, kotkas
3. Vesiroos, delfiin, ahven

5.2.3 Algkontsepti loomine

Järgnevalt leitakse iga klastri kohta üks kontsept, mille põhjal hakatakse leidma kontseptiahelat. Algkontsept leitakse ühe objekti põhjal, mis võetakse eelnevalt leitud klastrist. Objekti valimise kriteeriumiks on vähim atribuutide arv, sest sellisel juhul on suurem tõenäosus, et leitud kontsept asub võre ülemises osas ning see tagab omakorda suurema kontseptiahela katvuse. Suurem kontseptiahela katvus on tingitud asjaolust, et rohkem erinevaid tipmisi kontsepte satub erinevatesse kontseptiahelatesse. Tipmised kontseptid omavad kõige enam objektide ja atribuutide vahelisi relatsioone ning seega mõjutavad katvust ka kõige enam. Ühe kontseptiahela leidmise puhul see tähtsust ei oma, kuid mitme kontseptiahela puhul võib katvuse vahe olla märgatav. Konteksti katvuse erinevus, mis on tingitud algkontsepti asukoha valimisest, on näidatud järgnevas peatükis.

Igast eelmises peatükis leitud klastrist valitakse üks objekt, milleks on antud näite puhul: tamm, kotkas, ahven. Seejärel luuakse iga objekti põhjal üks algkontsept. Algkontsepti intentsiooniks on kõik objekti atribuudid ning kontsepti loomisel peab meeles pidama, et kontsepti ekstensiooni kuuluvad need objektid, millel on olemas kõik algse objekti atribuudid, kuid võivad olla ka muud atribuudid. Objektide põhjal leitud algkontseptideks on:

1. ({Tamm}, {Vajab eluks vett, Taim, Elab maa peal})
2. ({Kotkas}, {Vajab eluks vett, Elab maa peal, Lind, Lendab})
3. ({Vesiroos, Delfiin, Ahven}, {Vajab eluks vett, Elab vees})

Käesoleva andmestiku puhul asub 2 kontsepti võre alumises osas ning 1 kontsept võre ülemises osas.

5.2.4 Kontseptiahela leidmine

Iga eelmises punktis leitud algkontsepti kohta leitakse üks kontseptiahel. Algkontsept asub kontseptivõres esimese ning viimase kontsepti vahel. Selleks, et oleks võimalik ühe kontseptiahela leidmise algoritmi võimalikult palju taaskasutada, leitakse kontseptiahel kahes osas: esmalt leitakse kontseptiahel algkontseptist võre esimese kontseptini ning seejärel leitakse kontseptiahel võre viimasest kontseptist algkontseptini.

Enne kontseptiahela leidmist tuleb kontekstist valida ainult need objektid, mis võivad leitavas kontseptiahelas esineda. Algkontseptist ülespoole viivas kontseptiahelas võivad esineda vaid need objektid, millel on olemas vähemalt üks algkontsepti intentsioonis esinev atribuut. Senimaani näitena kasutatud loomade konteksti puhul on atribuut ’Vajab eluks vett’ ühine kõigi objektide jaoks, seega iga algkontsepti puhul esinevad ülespoole viivas kontseptiahelas kõik objektid.

Kasutades *numpy* poolt pakutavaid massiive, saab need objektid pärida ühe *numpy* käsuga, mis tagastab kõik objektide indeksid, millel on vähemalt üks nõutud atribuut olemas:

```
import numpy as np
np.argwhere(np.any(data[:, list(attributes)], axis=1))
```

Muutuja *attributes* on massiiv, mis hoiab endas kõiki algkontsepti atribuute. Muutuja *data* hoiab endas konteksti objektide ning atribuutide relatsioonide maatriksit, milleks on antud juhul:

$$\begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

Loomade konteksti puhul on mõnevõrra huvitavam leida neid objekte, mis asuvad allapoole viivas kontseptiahelas, sest sellisel juhul ei leita enam kõiki konteksti objekte. Nimelt nüüd tuleb valida vaid need objektid, millel on olemas kõik algkontsepti intentsiooni atribuudid. Kuid alamkontsepti definitsiooni järgi vastavad sellele tingimusele ainult need objektid, mis kuuluvad algkontsepti ekstensiooni ning neid eraldi leidma ei pea.

Seejärel on vaja leitud sobilikud objektid järjestada selliselt, et eespool oleksid tähtsamad objektid ehk need objektid, millel on kõige populaarsemad atribuudid. Algoritm itereerib kontseptiahela leidmisel üle kõigi objektide ning kui eespool on tähtsamad objektid, siis luues kontseptiahelaid on suurem tõenäosus, et tähtsamad

objektid satuvad kontseptiahelasse varem, mis omakorda tagab konteksti suurema katvuse kontseptiahela poolt. Objektide järjestamiseks kasutatakse miinustehnikat.

Kui objektid on tähtsuse järgi järjestatud, siis alustatakse kontseptiahela otsimisega. Kui esialgse kontseptiahela algoritmiga sai terve kontseptiahela leitud ühe korraga, siis magistritöö poolt pakutud lahenduse puhul tuleb ühe kontseptiahela leidmiseks rakendada algoritmi kaks korda – üks kord, et leida kontseptiahel kontseptivõre esimest kontseptist algkontseptini ning teine kord, et leida kontseptiahel algkontseptist võre viimase kontseptini. Samuti tuleb esialgset algoritmi veidi muuta, et ta oskaks arvestada algkontseptiga. Nimelt leidmaks kontseptiahelat algkontseptist võre viimase kontseptini rakendatakse olemasolevat algoritmi, kuid algoritm peatub, kui ta jõuab algkontseptini. Leidmaks kontseptiahelat võre esimesest kontseptist algkontseptini ei alustata kontseptiahela leidmist võre viimasest kontseptist, vaid algkontseptist ning algoritm peatub, kui kõik objektid on läbi käidud.

Kui mõlemad kontseptiahelad on leitud, siis nad ühendatakse ning moodustatakse terviklik kontseptiahel, mis viib kontseptivõre esimesest kontseptist kontseptivõre viimase kontseptini läbides algkontsepti. Eelmises peatükis leitud algkontseptide põhjal leitakse järgnevad kontseptiahelad:

({pingviin, kotkas, inimene, koer, karu, tamm, vesiroos, delfiin, ahven}, {vajab eluks vett}) →
({pingviin, kotkas, inimene, koer, karu, tamm}, {vajab eluks vett, elab maa peal}) →
({tamm}, {vajab eluks vett, taim, elab maa peal}) →
({}, {vajab eluks vett, taim, elab maa peal, elab vees, imetaja, lind, lendab, liigub kahel jalal}})

({pingviin, kotkas, inimene, koer, karu, tamm, vesiroos, delfiin, ahven}, {vajab eluks vett}) →
({vesiroos, delfiin, ahven}, {vajab eluks vett, elab vees}) →
({delfiin}, {vajab eluks vett, elab vees, imetaja}) →
({}, {vajab eluks vett, taim, elab maa peal, elab vees, imetaja, lind, lendab, liigub kahel jalal}

({pingviin, kotkas, inimene, koer, karu, tamm, vesiroos, delfiin, ahven}, {vajab eluks vett}) →

({pingviin, kotkas, inimene, koer, karu, tamm}, {vajab eluks vett, elab maa peal}) →
({pingviin, kotkas}, {vajab eluks vett, elab maa peal, lind}) →
({kotkas}, {vajab eluks vett, elab maa peal, lind, lendab}) →
({}, {vajab eluks vett, taim, elab maa peal, elab vees, imetaja, lind, lendab, liigub kahel jalal})}}

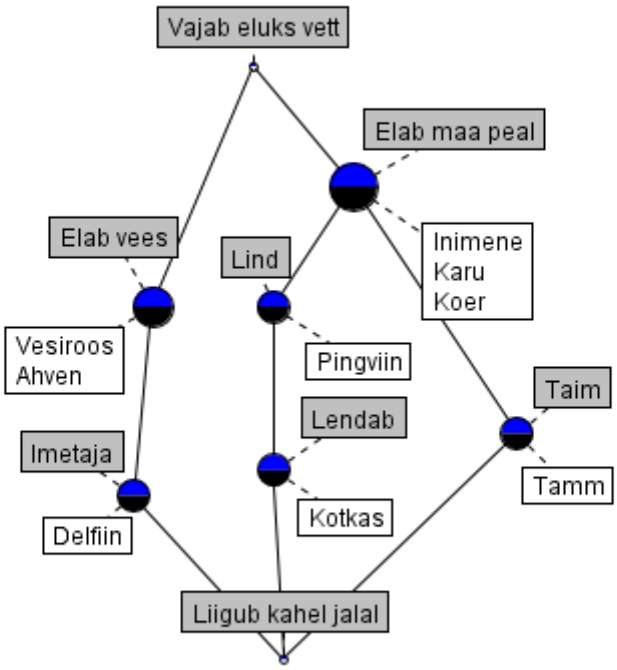
Kolm leitud kontseptiahelat katavad ära 79% kontekstist.

5.2.5 Kontseptiahela visualiseerimine

Leitud kontseptivõrede visualiseerimisel kasutatakse sama kontseptide ekstensioonide ning intentsioonide esitamise põhimõtet nagu kontseptivõrede visualiseerimisel. Erinevus seisneb vaid selles, et kontseptiahela puhul on igal kontseptil vaid üks alamkontsept. Samuti võib esitada kõiki kontseptiahelaid korraga moodustades neist uue kontseptivõre, mis on loodud uue konteksti põhjal. Uueks kontekstiks on need objektide ja atribuutide relatsioonid, mis on kontseptiahelate poolt kaetud.

Mõlemal juhul kasutatakse kontseptiahelate visualiseerimiseks ConExp tööriista, sest see pakub lisaks visualiseerimisele ka muid kontseptivõredega töötamise võimalusi. Algoritm väljastab uue vähendatud konteksti CSV failina, mida saab kasutada ConExp'i sisendina.

Joonis 6 kujutab loomade konteksti kolme leitud kontseptiahelat.



Joonis 6. Loomade konteksti kontseptiahelad kontseptivõrena

6 Analüüs

Käesolevas peatükis analüüsitakse loodud algoritmi rakendades seda loomaaia konteksti peal, mida kasutati töö esimeses osas, et illustreerida kuivõrd halvasti on suured kontseptivõred loetavad. Selle konteksti põhjal näidatakse, kuidas kontseptiahelate leidmisel konteksti suurus kasvab ning seletatakse, millest katvuse kasv on tingitud. Samuti hinnatakse kontseptiahelate katvust suurte, kümnetest tuhandetest kontseptidest koosnevate, kontekstide peal. Samuti räägitakse algoritmi jõudlusest ning millised moel seda parandada saab. Samuti võrreldakse kontseptiahelate meetodit jäämäe kontseptivõredega. Ühtlasi võrreldakse loodud lahendust olemasoleva lahendusega.

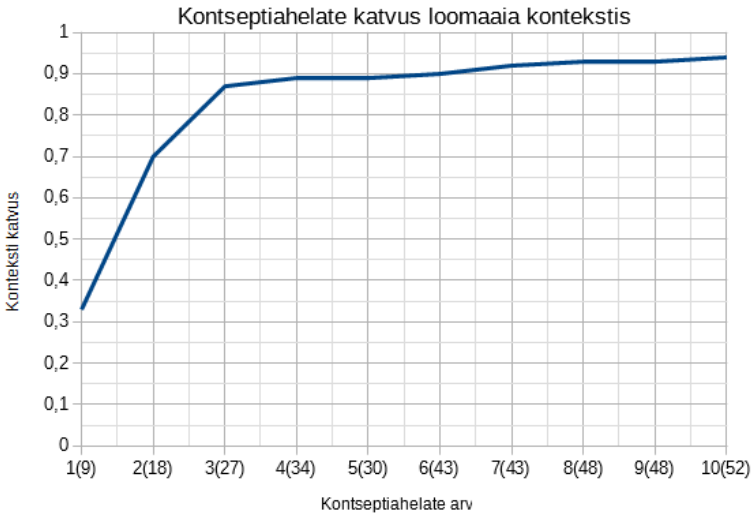
6.1 Algoritmi rakendamine suurele kontekstile

Lisa 2-s toodud loomaaia kontekstis on 238 erinevat kontsepti ning neist moodustuv kontseptivõre on väga raskesti hoomatav. Käesolevas peatükis rakendatakse loodud algoritmi loomaaia kontekstile ning saadakse teada, kuivõrd hästi mitme kontseptiahela meetod seda konteksti iseloomustab.

Joonis 7-l kujutatud graafikul on näha, kuidas kontseptiahelate arvu kasvades suureneb ka konteksti katvus. Kontseptiahelate arvu järel sulgudes olev arv tähistab unikaalsete kontseptide arvu kõigis kontseptiahelates. Väiksema kontseptiahelate arvu korral kasvab katvus võrdlemisi kiiresti, kuid siis järsk katvuse kasv aeglustub. Põhjus seisneb selles, et esimesed kontseptiahelad katavad ära kontseptivõre tipmised kontseptid ning sellega kaasneb ka suurem katvuse kasv. Kui kontseptiahelate arv kasvab, siis uued kontseptiahelad sisaldavad juba tipmisi kontsepte ning seetõttu enam suurt katvuse suurenemist ei toimu. Suurema kontseptiahelate arvu korral lisanduvad uued kontseptid kontseptivõre madalamatest osadest ning seega katvuse protsent muutub vähe. Sajaprotsendiline katvus saavutatakse 30 kontseptiahelaga leides kokku 99 erinevat kontsepti.

Kuna kontseptiahelate leidmine on seotud tihedalt klasterdamisega, siis erineva kontseptiahelate arvu korral leitakse erinevad klastrid ning nende klastrite põhjal leitud algkontseptid võivad erineda teistsuguse kontseptiahelate hulga puhul leitud algkontseptidest. Sellest tulenevalt võib esineda olukordi, kus suurendades näiteks kontseptiahelate arvu ühe võrra ei pruugi kõik eelnevalt leitud kontseptiahelad uues

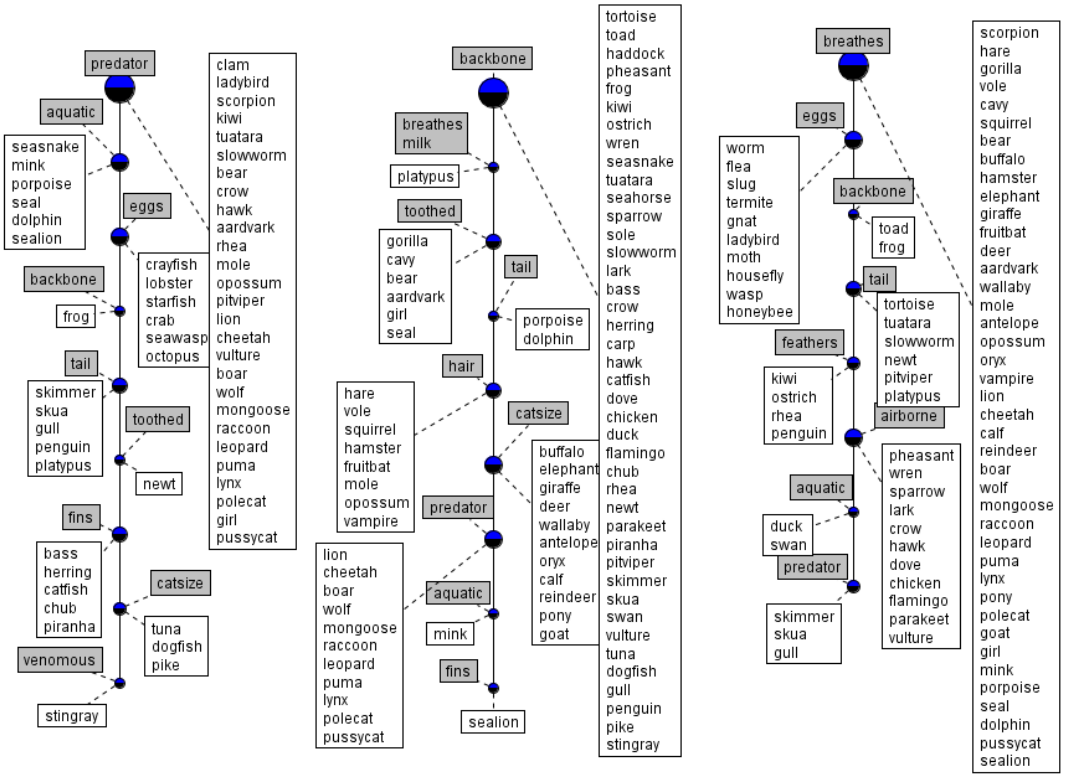
kontseptiahelate hulgas esineda. See põhjendab ka graafikul erinevust 6 ja 7 kontseptiahela vahel, kus katvuse protsent veidi muutub, kuid unikaalsete kontseptide arv jääb samaks. Samal põhjusel on 8 ja 9 kontseptiahela puhul katvuse protsent ning unikaalsete kontseptide arv sama. Sellist algoritmi käitumist võib pidada käesoleva lahenduse miinuseks.



Joonis 7. Kontseptiahelate katvus loomaaia kontekstis

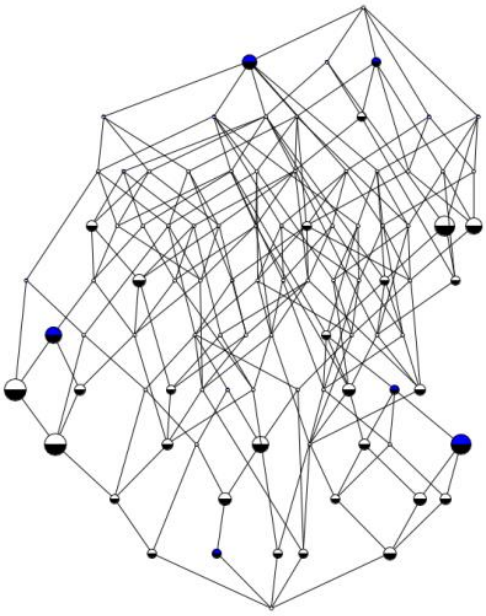
Kui eelmises peatükis toodud loomade konteksti näites ei olnud konteksti väikese suuruse tõttu vahet, kas valida algkontsept kontseptivõre ülemisest osast või kontseptivõre alumisest osast, siis suurema loomaaia konteksti puhul on konteksti katvuse erinevus vägagi märgatav. Ühe kontseptiahela puhul ei ole algkontsepti asukoht tähtis, kuid leides mitu kontseptiahelat mängib algkontsepti valik väga suurt rolli. Näiteks kahe kontseptiahela puhul valides algkontseptid kontseptivõre ülemisest osast on konteksti katvus 70%, samas valides algkontseptid kontseptivõre alumisest osast on katvus kõigest 50%. Kolme kontseptiahela puhul on katvus vastavalt 87% ja 62%. Sellistest tulemustest tulenevalt võib väita, et algkontsepti valimine kontseptivõre ülemisest osast on parem.

Joonis 8-l on kujutatud 3 kontseptiahelat, mis on leitud loomaaia kontekstist.



Joonis 8. Kolm kontseptiahelat loomaaia kontekstis

Joonis 9-l on kujutatud vähendatud kontekst mitte kontseptiahelate kujul, vaid kontseptivõrena. Põhjus, miks kontseptivõrena esitatud kontseptiahelad sisaldavad märgatavalt rohkem kontsepte kui on kujutatud kontseptiahelates Joonis 8-l, seisneb selles, et kontseptivõres on kujutatud lisaks kontseptiahelas olevatele kotnseptidele ka need kontseptid, mis tekivad kontseptiahelate lõikumisel. Ning kuigi kontseptivõres on kontsepte rohkem kui kontseptiahelas, siis sellegipoolest on uus kontseptivõre võrreldes esialgse kontseptivõrega, mis on kujutatud Joonis 2-l, palju loetavam ning selgem, kuid sellegipoolest katab ära väga suure osa kontekstist.



Joonis 9. Loomaaia kolme kontseptiahela kontseptivõre

6.2 Algoritmi jõudlus

Mitme kontseptiahela leidmise algoritmi keerukus on $O(n^2)$, sest andmete klasterdamise, algkontsepti leidmise ning kontseptiahela leidmise keerukus on $O(n)$, kuid miinustehnika rakendamise keerukus on $O(n^2)$. Lähtuvalt sellest oleks mõistlik miinustehnikat mitte kasutada ning järjestada objektid tähtsuse järgi nii, et keerukus oleks väiksem. Kuna miinustehnika kasutab ühe sammuna konformismiskaalat ehk iga objekti kaalu arvutamist, siis järgnevalt võrreldakse, kas ainult konformismiskaala abil on võimalik saavutada miinustehnikaga sarnane konteksti katvus. Konformismiskaala leidmise keerukus on $O(n)$.

Konformismiskaala rakendamisel on konteksti katvus vaid paari protsendi võrra väiksem kui miinustehnika puhul, kuid kontseptiahelate leidmiseks kuluv aeg on märgatavalt väiksem. Näiteks katsetades kontekstiga, milles on 20 000 objekti kulus miinustehnika rakendamiseks umbes minut ning konformismiskaala võttis alla poole sekundi. Kui miinustehnika kasutamisest saadud suurem konteksti katvus on väga oluline, siis võib seda kasutada niikaua kui kontekstis on kuni paarkümmend tuhat objekti, kuid suurema objektide arvu korral on mõistlikum juba kasutada konformismiskaalat.

6.3 Algoritmi rakendamine erinevate andmestikega

Järgnevalt rakendatakse algoritmi erinevatele andmestikele, et näha, kuidas käitub algoritm andmestikega, kus on erinev hulk objekte ning atribuute ning kas on olemas andmestikke, mille puhul antud töös pakutud lahenduse rakendamine on vähem efektiivne ning millised andmestiku omadused võivad konteksti katvust mõjutada. Kõik andmestikud on oma olemuselt kategoorilised ehk igal atribuudil on mingi kindel hulk võimalikke väärtusi. Selleks, et neid saaks kasutada formaalses kontseptianalüüsis peab nad teisendama binaarsele kujule. Näiteks, kui atribuudil on kolm võimalikku väärtust, siis tehakse igast võimalikust atribuudi väärtusest uus atribuut, mille väärtuseks saab olema kas 0 või 1. Teisendamiseks kasutati *pandase* teeki:

```
import pandas as pd
data = pd.read_csv(„in.csv", sep='\s*,\s*', index_col=False)
pd.get_dummies(data).to_csv('out.csv')
```

Kõik andmestikud pärinevad UCI masinõppe repositooriumist [10]. Andmestike tausta ning objektide ja atribuutide sisu lähemalt ei tutvustata, kuid andmestikke iseloomustavad metaandmed tuuakse välja Tabel 5-s. Andmestiku tihedus tähistab, kui suur osa kõigist võimalikest objektide ning atribuutide vahelistest relatsioonidest on tabelis kujutatud. Samuti on antud andmestikud heaks näiteks sellest, kuivõrd kiiresti kontseptide arv võrreldes objektide arvuga kasvab.

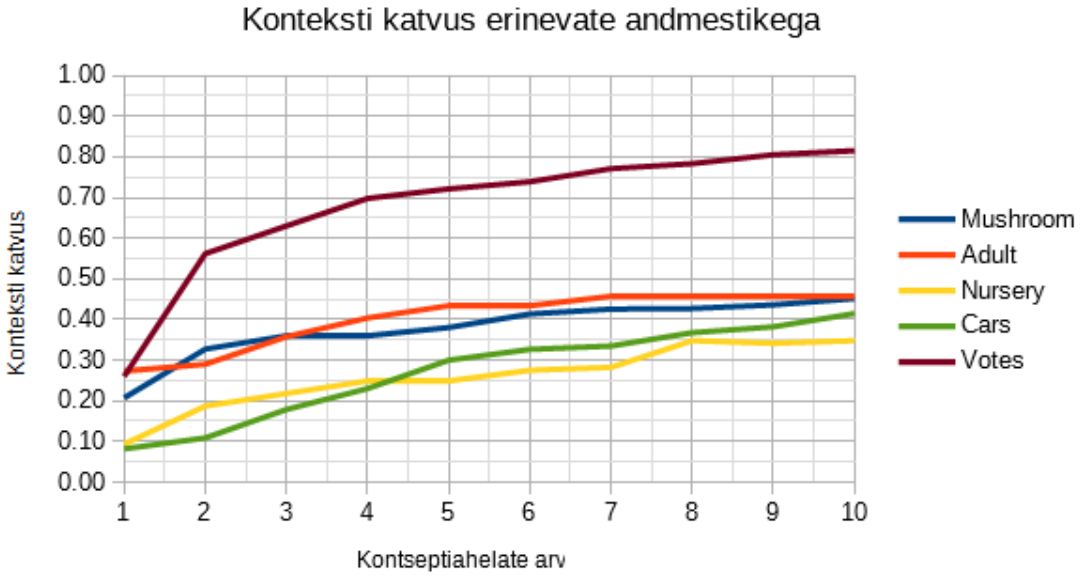
Andmestik	Objektide arv	Atribuutide arv	Tihedus	Kontseptide arv
Mushroom	8124	117	19,64%	157 138
Adult	36 561	102	7,99%	86 386
Nursery	12 960	27	29,63%	115 201
Car evaluation	1728	21	28,57%	8001
Congressional voting records	435	17	52.14%	10 644

Tabel 5. Kasutatavate andmestike metaandmed

Joonis 10-l kujutatud graafikul on näha, kuidas konteksti katvus erinevate andmestikega muutub, kui kontseptiahelate arv suureneb. Kõigis viies andmestikus on kontseptiahelad leitud kasutades konformismiskaalat, sest objektide arvud on miinustehnika jaoks mõne andmestiku puhul üpris suured. Kõigi andmestike ühiseks jooneks on konteksti katvuse märgatav tõus esimeste kontseptiahelate puhul. Tulemusi lähemalt uurides selgub, et konteksti katvuse erinevus ei ole põhjustatud andmestiku tihedusest ega ka kontseptide arvust.

Põhjus, miks erinevate kontekstide katvus on erinev, seisneb andmete omavahelistes seostes ning nende kontseptivõrede struktuuris. Nimelt, kui kontekstis olevad objektid on teineteisest erinevad, siis on võre laiem ning madalam. Selle tõttu on ka kontseptiahelate katvus madalam. Seda iseloomustab ilmekalt viimane andmestik (*Congressional voting records*), kus käsitletakse kahe erineva erakonna esindajate hääletamisi. Kuna sama erakonna esindajad kipuvad hääletama sarnaselt, siis on selle andmestiku objektid omavahel sarnasemad kui teistes andmestikes. Seetõttu on ka konteksti katvus teiste andmestikega võrreldes märksa suurem.

Kokkuvõtvalt võib täie kindlusega väita, et rohkem kui ühe kontseptiahela leidmine toob kaasa ka suurema konteksti katvuse.

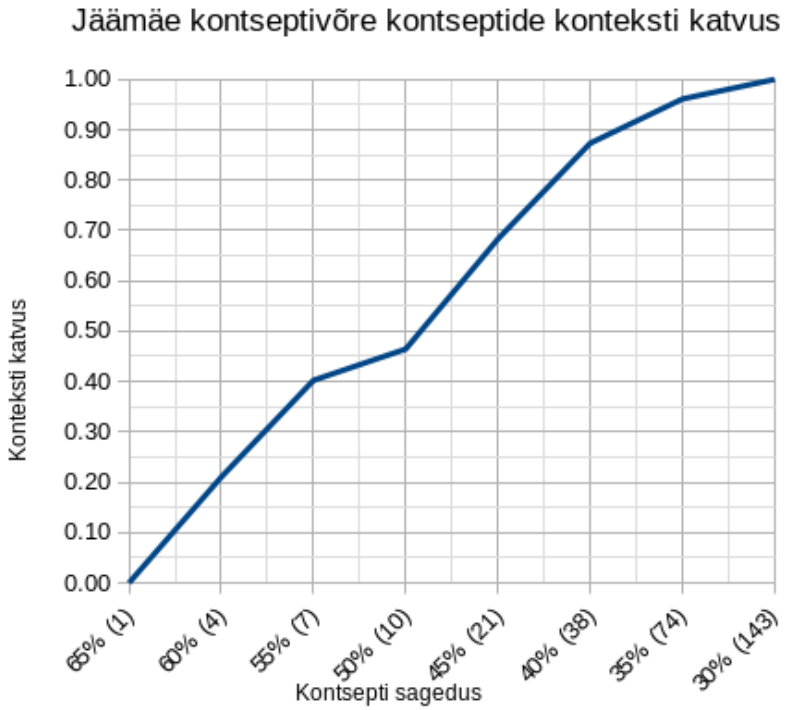


Joonis 10. Konteksti katvus erinevate andmestikega

6.4 Võrdlus jäämäe tüüpi kontseptivõrega

Kolmandas peatükis tutvustati kontseptivõrede lihtsustamist jäämäe tüüpi kontseptivõrede meetodi abil. Selle lähenemise puhul näidatakse ainult kontseptivõre ülemises osas olevaid kontsepte. Võrreldakse just selle meetodiga, sest kõigi kolmandas peatükis vaadeldud kontseptivõrede lihtsustamise meetodite seast on kontseptiahelate meetodiga enim sarnane just jäämäe tüüpi kontseptivõrede meetod.

Järgnevalt võrreldakse kontseptiahelate leidmise meetodit jäämäe tüüpi kontseptivõrega. Võrdlust teostatakse eelmises peatükis kasutatud *Congressional voting records* andmestiku põhjal. Joonis 11-l on näidatud, kuidas kasutades jäämäe kontseptivõresid muutub konteksti katvus, kui horisontaalteljel kuvatud kontsepti sageduse piirväärtus muutub. Sulgudes on kuvatud leitud kontseptide arv. Piirväärtus on kahanevas järjestuses, sest kõige sagedasemad kontseptid on kontseptivõre tipus ning kui kontsepti sagedus on suur, siis järelikult asub kontsept kontseptivõre tipus. Sageduse vähenedes valitakse kontseptivõre alumises osas asuvad spetsiifilisemad kontseptid ning leitud kontseptide arv suureneb.



Joonis 11. Jäämäe kontseptivõre kontseptide konteksti katvus

Nagu eelmises peatükis selgus, siis selle andmestiku puhul on kontseptiahelate katvus üpris suur. Nimelt juba nelja kontseptiahela puhul, milles on kokku 43 unikaalset

kontsepti, on konteksti katvus 70%. Nagu Joonis 11-l näha on, siis jäämäe tüüpi kontseptivõrede puhul saavutatakse ligilähedane (68%) katvus leides 21 tipmist kontsepti. Üldjuhul saavutatakse jäämäe kontseptivõredega suurem konteksti katvus ning suure katvuse jaoks on vaja ka vähem kontsepte. See on tingitud asjaolust, et jäämäe kontseptivõre puhul näidatakse ainult tipmisi kontsepte, mis katavad kõige suurema osa kontekstist, sest nendel kontseptidel on palju objekte.

Esmapilgul võib tunduda, et kui kontseptiahelate puhul on vaja sama konteksti katvuse jaoks leida kaks korda rohkem kontsepte kui jäämäe kontseptivõre puhul, siis on kontseptiahelate meetod seetõttu halvem. Tuleb silmas pidada, et kontseptiahelate ning jäämäe tüüpi kontseptivõrede erinevuseks on see, millised kontsepte leitakse. Nimelt jäämäe tüüpi kontseptivõrede puhul näidatakse ainult tipmisi kontsepte ning seega antakse ülevaade ainult ülemistest kontseptidest. Kontseptiahelate puhul leitakse kontseptid, mis läbivad vertikaalselt kogu kontseptivõret ning sisaldavad seetõttu ka spetsiifilisemaid kontsepte. Kokkuvõtvalt võib väita, et mitme kontseptiahela leidmise meetod on võrreldes jäämäe tüüpi kontseptivõredega üpriski hea, sest selle abil on võimalik konteksti kirjeldada efektiivselt väikese kontseptide arvuga.

6.5 Võrdlus esialgse lahendusega

Magistritöö eesmärgiks oli muuta esialgset kontseptiahelate leidmise lahendust nõnda, et oleks võimalik leida rohkem kui ühte kontseptiahelat, kuid lisaks sellele sai tehtud veel mitmeid parendusi.

Üheks püstitatud kõrvaleesmärgiks oli algoritmi optimeerida. Magistritöö käigus leitigi esialgses lahenduses mõned kitsaskohad, mille kõrvaldamisel muutus algoritm märgatavalt kiiremaks. Nimelt oli probleem miinustehnika arvutamises, kus esialgselt ei kasutatud kõiki raamistike poolt pakutavaid võimalusi andmetabelitega töötamiseks ning tehti üleliigseid arvutusi, mis tegid kogu protsessi aeglasemaks.

Võrreldes esialgse lahendusega on algoritmi mugavam kasutada. Samuti on kasutatavust parandatud kogu koodile testide kirjutamisega, mis katavad ära enam kui 95% koodirida ning tänu sellele saab olemasolevat koodi tulevikus lihtsamini muuta ning suureneb ka tõenäosus, et kood töötab korrektselt. Samuti muudab see teistel isikutel lihtsamini koodist aru saada ning seda ise arendama hakata.

7 Kokkuvõte

Magistritöös anti ülevaade formaalse kontseptianalüüsi põhimõistete kohta ning näidati, kuidas saab formaalseid kontsepte kontseptivõrede abil visualiseerida. Samuti kirjeldati probleemi kontseptivõrede visualiseerimisega, kus suurema kontseptide arvuga kontseptivõred muutuvad inimsilma jaoks väga kiiresti loetamatuks. Töös anti ülevaade mitmest erinevast olemasolevast lähenemisest, mille abil saab kontseptide või kontekstis olevate objektide ja atribuutide vähendamisega muuta kontseptivõred loetavamaks ning hoomatavamaks, kuid mitte ükski praeguseks välja pakutud lahendusest ei ole veel muutunud selles valdkonnas standardiks, mis kõigile sobiks. Seetõttu oli mõistlik kontseptivõrede esitamise probleemiga tegeleda.

Üheks kontseptivõrede lihtsustamise võimaluseks on leida kontseptivõrest kontseptiahel, mis on tee kontseptivõre esimesest kontseptist kontseptivõre viimase kontseptini. Samas ühe kontseptiahela leidmine ei kirjelda kogu uuritavat andmetabelit piisavalt hästi ning oleks parem, kui saaks kontseptivõres leida mitu kontseptiahelat. Algoritm, mille abil saab leida ühte kontseptiahelat, oli juba olemas. Magistritöö põhieesmärgiks oli arendada edasi olemasolevat lahendust nõnda, et oleks võimalik leida mitut kontseptiahelat korraga. Magistritöö tulemusena valminud algoritm arendas edasi esialgse algoritmi ideed, kus kontseptiahelat leiti kogu kontseptivõret genereerimata.

Selleks, et saaks hinnata, kuivõrd palju on mitme kontseptiahela leidmine ühe kontseptiahela leidmisest parem, kasutati konteksti katvuse mõõdikut. Konteksti katvus näitab, kui palju objektide ning atribuutide vahelisi relatsioone on kontseptiahelate poolt kaetud. Sama mõõdikut kasutati ka loodud algoritmi hindamiseks erinevate suure kontseptide arvuga kontekstide peal, et näha, kuidas kontseptiahelate arvu suurendamisel muutub konteksti katvus. Samuti selgus töö käigus, et miinustehnika kasutamine väga suurt kasu ei too ning suurte kontekstide puhul on võimalik kasutada objektide järjestamiseks ka konformismiskaalat, mille puhul on konteksti katvus mõnevõrra väiksem, kuid samas leitakse kontseptiahelad märgatavalt kiiremini. Samuti võrreldi konteksti katvust kasutades mitme kontseptiahela ning jäämäe tüüpi kontseptivõrede meetodeid.

Mitme kontseptiahela leidmisel erinevate andmestikega on näha, et konteksti katvus on suurem kui ainult ühe kontseptiahela puhul. Seega töö hüpotees, mis väitis, et rohkem kui ühe kontseptiahela leidmisel on konteksti katvus suurem kui ainult ühe kontseptiahela leidmisel, peab paika.

7.1 Võimalikud edasiarendused

Hetkel kasutatakse algkontseptide leidmiseks klasterdamist. Selle lähenemise miinuseks on asjaolu, et algkontseptid valitakse põhimõtteliselt juhuslikult (kuid deterministlikult) ning algoritmi kasutajal ei ole kontrolli selle üle, milline kontsept osutub algkontseptiks. Selliselt valitud algkontseptid ei pruugi olla kõige optimaalsemad ning see võib põhjustada olukorra, kus veel ühe kontseptiahela leidmine vähendab konteksti katvust, mitte ei suurenda seda. Põhjus seisneb selles, et erineva kontseptiahelate arvu korral leitakse hoopis uued klastrid, seeläbi ka täiesti uued algkontseptid ning ka täiesti uued kontseptiahelad. Kui näiteks leida mõnel kontekstil alguses neli kontseptiahelat ning seejärel leida viis kontseptiahelat, siis ei pruugi viie kontseptiahela hulgas olla ühtegi kontseptiahelat esimesena leitud nelja kontseptiahela hulgas. Samuti võib juhtuda, et algkontseptid on teineteisele liiga lähedal ning leitud kontseptiahelad hakkavad omavahel kattuma.

Üheks edasiarenduse eesmärgiks oleks leida uus lähenemine, kus algkontseptid valitakse mingi kindla kriteeriumi järgi. Üheks võimalikuks kriteeriumiks oleks genereerida kontseptivõre ülemine osa ning sealt leida algkontseptid. Sel moel saaks tagada, et leitavad kontseptiahelad ei hakka omavahel väga palju kattuma ning konteksti katvus uute kontseptiahelate leidmisel ei kahane.

Kuigi kontseptiahelate leidmist võrreldi jäämäe kontseptivõredega, siis sellegipoolest tuleks kontseptiahela meetodit rakendada andmeteaduses andmestike analüüsimiseks ning reaalse probleemi lahendamiseks, sest selles töös analüüsiti leitud kontseptiahelaid päris andmetel üpriski pinnapealselt. Oleks huvitav teada, kas kontseptiahelate meetodit on otstarbekas kasutada ning kuivõrd hästi suudab see konkureerida teiste kontseptivõre lihtsustamise meetoditega.

Kasutatud kirjandus

[1] U. Priss, "Formal Concept Analysis in Information Science," *Annual Review of Information Science and Technology,* pp. 521-543, 2006.

[2] B. Ganter ja R. Wille, Formal Concept Analysis, Mathematical Foundations, Springer, 1999.

[3] R. Wille, „Formal Concept Analysis," %1 *Formal Concept Analysis As Mathematical Theory of Concepts and Concept Hierarchies,* pp. 1-33.

[4] A. Torim, *Formal Concepts in the Theory of Monotone Systems,* Tallinn: Tallinn University of Technology, 2009.

[5] M. Klimushkin, S. Obiedkov ja C. Roth, „Approaches to the selection of relevant concepts in the case of noisy data," %1 *International Conference on Formal Concept Analysis,* 2010.

[6] N. Mimouni, A. Nazarenko ja S. Salotti, „A Conceptual Approach for Relational IR: Application to Legal Collections," 2015.

[7] A. Castellanos, A. García-Serrano ja J. Cigarrán, „Linked Data-based Conceptual Modelling for Recommendation: A FCA-Based Approach," 2014.

[8] G. J. Greene, M. Esterhuizen ja B. Fischer, „lattices, Visualizing and exploring software version control repositories using interactive tag clouds over formal concept," *Information and Software Technology,* kd. 87, pp. 223-241, 2017.

[9] T. Quan, S. Cheung Hui ja T. Hoang Cao, „A Fuzzy FCA-based Approach to Conceptual Clustering for Automatic Generation of Concept Hierarchy on Uncertainty Data.," 2004.

[10] D. Dheeru ja E. Karra Taniskidou, *{UCI} Machine Learning Repository,* University of California, Irvine, School of Information and Computer Sciences, 2017.

[11] S. M. Dias ja N. J. Vieira, „Concept lattices reduction: Definition, analysis and classification," *Expert Systems with Applications,* kd. 42, pp. 7084-7097, 2015.

[12] B. Grand, M. Soto ja M.-A. Aufaure, „Conceptual and Spatial Footprints for Complex Systems Analysis: Application to the Semantic Web," %1 *Proceedings of the 20th International Conference on Database and Expert Systems Applications,* Berlin, 2009.

[13] S. Dias ja N. Vieira, „Reducing the Size of Concept Lattices: The JBOS Approach.," %1 *CEUR Workshop Proceedings,* 2010.

[14] C. Aswani Kumar ja S. Srinivas, „Concept lattice reduction using fuzzy K-Means clustering," *Expert Systems with Applications,* kd. 37, nr 3, pp. 2696-2704, 2010.

[15] R. Belohlavek ja J. Macko, „Selecting Important Concepts Using Weights," %1 *International Conference on Formal Concept Analysis,* 2011.

[16] G. Stumme, R. Taouil, Y. Bastide, N. Pasquier ja L. Lakhal, „Computing iceberg concept lattices with Titanic," *Data and Knowledge Engineering,* pp. 189-222, August 2002.

[17] S. Andrews ja C. Orphanides, „Analysis of large data sets using formal concept lattices," %1 *Proceedings of the 7th International Conference on Concept Lattices and Their Applications,* Seville, 2010.

[18] R. Belohlavek ja V. Vychodil, „Closure-based constraints in formal concept

analysis,“ *Discrete Applied Mathematics,* p. 1894–1911, 2013.

[19] S. M. Dias ja N. J. Vieira, „A methodology for analysis of concept lattice reduction,“ *Information Sciences,* kd. 396, pp. 202-217, 2017.

[20] C. Melo, B. Le-Grand, M. A. Aufaure ja A. Bezerianos, „Extracting and Visualising Tree-like Structures from Concept Lattices,“ %1 *2011 15th International Conference on Information Visualisation*, 2011.

[21] A. Gely, „Links between Modular Decomposition of Concept Lattice and Bimodular Decomposition of a Context,“ 2011.

[22] T. Hastie, R. Tibshirani ja J. Friedman, „The Elements of Statistical Learning,“ 2008, pp. 509-510.

[23] Z. Huang, „Extensions to the k-Means Algorithm for Clustering,“ *Data Mining and Knowledge Discovery,* pp. 283-304, 1998.

[24] S. Amiri, B. Clarke ja J. Clarke, „Clustering Categorical Data via Ensembling Dissimilarity Matrices,“ *Journal of Computational and Graphical Statistics,* 2015.

[25] Z. He, X. Xu ja S. Deng, „Attribute Value Weighting in K-modes Clustering,“ *Expert Systems with Applications,* 2011.

[26] J. A. Hartigan ja M. A. Wong, „Algorithm AS 136: A K-Means Clustering Algorithm,“ *Journal of the Royal Statistical Society,* pp. 100-108, 1979.

[27] J. E. Cornell, J. A. Pugh, J. W. Williams, L. Kazis, A. F. Lee, M. L. Parchman, J. Zeber, T. Pederson, K. A. Montgomery ja P. H. Noël, „Multimorbidity Clusters: Clustering Binary Data From a Large Administrative Medical Database,“ *Applied Multivariate Research,* kd. 12, pp. 163-182, 2007.

[28] O. Maimon ja L. Rokach, „Clustering Methods,“ %1 *Data Mining and Knowledge Discovery Handbook,* p. 279.

[29] E. Martin, H.-P. Kriegel, J. Sander ja X. Xu, „A Density-Based Algorithm for Discovering Clusters,“ %1 *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, 1996.

[30] L. Vohandu, R. Kuusik, A. Torim, E. Aab ja G. Lind, „Some algorithms for data table (re)ordering using Monotone Systems,“ 2006.

Lisa 1 – Programmikood

Magistritöö käigus kirjutatud kood on leitav Git'i versioonihalduse süsteemist:

https://bitbucket.org/metsmarko/fca-chain/

Lisa 2 – Loomaaia kontekst

	hair	feathers	eggs	milk	airborne	aquatic	predator	toothed	backbone	breathes	venomous	fins	tail	domestic	cat size
aardvark	X			X			X	X	X	X					X
antelope	X			X				X	X	X			X		X
bass			X			X	X	X	X			X	X		
bear	X			X			X	X	X	X					X
boar	X			X			X	X	X	X			X		X
buffalo	X			X				X	X	X			X		X
calf	X			X				X	X	X			X	X	X
carp			X			X		X	X			X	X	X	
catfish			X			X	X	X	X			X	X		
cavy	X			X				X	X	X				X	
cheetah	X			X			X	X	X	X			X		X
chicken		X	X		X				X	X			X	X	
chub			X			X	X	X	X			X	X		
clam			X				X								
crab			X			X	X								
crayfish			X			X	X								
crow		X	X		X		X		X	X			X		
deer	X			X				X	X	X			X		X
dogfish			X			X	X	X	X			X	X		X
dolphin				X		X	X	X	X	X		X	X		X
dove		X	X		X				X	X			X	X	
duck		X	X		X	X			X	X			X		
elephant	X			X				X	X	X			X		X
flamingo		X	X		X				X	X			X		X
flea			X							X					
frog			X			X	X	X	X	X					
fruit bat	X			X	X			X	X	X			X		
giraffe	X			X				X	X	X			X		X
girl	X			X			X	X	X	X				X	X
gnat			X		X					X					
goat	X			X				X	X	X			X	X	X
gorilla	X			X				X	X	X					X
gull		X	X		X	X	X		X	X			X		
haddock			X			X		X	X			X	X		
hamster	X			X				X	X	X			X	X	
hare	X			X				X	X	X			X		
hawk		X	X		X		X		X	X			X		
herring			X			X	X	X	X			X	X		
honeybee	X		X		X					X	X			X	
housefly	X		X		X					X					
kiwi		X	X				X		X	X			X		
ladybird			X		X		X			X					
lark		X	X		X				X	X			X		
leopard	X			X			X	X	X	X			X		X
lion	X			X			X	X	X	X			X		X
lobster			X			X	X								
lynx	X			X			X	X	X	X			X		X
mink	X			X		X	X	X	X	X			X		X
mole	X			X			X	X	X	X			X		

mongoose	X			X			X	X	X	X			X		X
moth	X		X		X					X					
newt			X			X	X	X	X	X			X		
octopus			X			X	X								X
opossum	X			X			X	X	X	X			X		
oryx	X			X				X	X	X			X		X
ostrich		X	X						X	X			X		X
parakeet		X	X		X				X	X			X	X	
penguin		X	X			X	X		X	X			X		X
pheasant		X	X		X				X	X			X		
pike			X			X	X	X	X			X	X		X
piranha			X			X	X	X	X			X	X		
pit viper			X				X	X	X	X	X		X		
platypus	X		X	X		X	X		X	X			X		X
polecat	X			X			X	X	X	X			X		X
pony	X			X				X	X	X			X	X	X
porpoise				X		X	X	X	X	X		X	X		X
puma	X			X			X	X	X	X			X		X
pussycat	X			X			X	X	X	X			X	X	X
raccoon	X			X			X	X	X	X			X		X
reindeer	X			X				X	X	X			X	X	X
rhea		X	X				X		X	X			X		X
scorpion							X			X	X		X		
seahorse			X			X		X	X			X	X		
seal	X			X		X	X	X	X	X		X			X
sea lion	X			X		X	X	X	X	X		X	X		X
sea snake						X	X	X	X		X		X		
sea wasp			X			X	X				X				
skimmer		X	X		X	X	X		X	X			X		
skua		X	X		X	X	X		X	X			X		
slowworm			X				X	X	X	X			X		
slug			X							X					
sole			X			X		X	X			X	X		
sparrow		X	X		X				X	X			X		
squirrel	X			X				X	X	X			X		
starfish			X			X	X								
stingray			X			X	X	X	X		X	X	X		X
swan		X	X		X	X			X	X			X		X
termite			X							X					
toad			X			X		X	X	X					
tortoise			X						X	X			X		X
tuatara			X				X	X	X	X			X		
tuna			X			X	X	X	X			X	X		X
vampire	X			X	X			X	X	X			X		
vole	X			X				X	X	X			X		
vulture		X	X		X		X		X	X			X		X
wallaby	X			X				X	X	X			X		X
wasp	X		X		X					X	X				
wolf	X			X			X	X	X	X			X		X
worm			X							X					
wren		X	X		X				X	X			X		