TARTU ÜLIKOOL
Arvutiteaduse instituut
Informaatika õppekava

Joonas Puura

Tarkvara loomine erinevate k-keskmiste algoritmide rakendamiseks

Bakalaureusetöö (9 EAP)

Juhendaja: Jaak Vilo, PhD

Tartu 2016

Tarkvara loomine erinevate k-keskmiste algoritmide rakendamiseks

Lühikokkuvõte:

Klasteranalüüsis on laialt levinud k-keskmiste meetod, mis võimaldab andmeid grupeerida nende tunnuste järgi, seejuures minimeerides ruutvigade summat klastrites olevate andmeobjektide ja vastava klastri keskpunktide vahel. Kuna k-keskmiste meetodi kui optimeerimisülesandele täpse lahenduse leidmine on NP-raske, siis on probleemi lahendamiseks võetud kasutusele mitmeid lähendeid otsivaid algoritme. Bakalaureusetöö eesmärgina valmis rakendus, mis lubab kasutada viit k-keskmiste klasterdusalgoritmi ja nelja algsete keskpunktide valimise meetodit. Kasutades nii reaalelulisi kui ka sünteetilisi andmestikke antakse ülevaade rakenduses implementeeritud algoritmide jõudlusest, mälukasutusest ja edukusest leida hea lähend k-keskmiste optimeerimisülesandele.

Võtmesõnad:

k-keskmised, klasteranalüüs, algoritmid

CERCS: P175 Informaatika, süsteemiteooria

Software for clustering using k-means algorithms

Abstract:

In cluster analysis k-means method is a method popularly used for grouping data by their features. The method aims to minimize within-cluster sum of squared errors between data objects in clusters and their corresponding center means. Because solving k-means optimization task exactly is NP-hard there have been introduced several heuristic algorithms for finding approximations. As the goal of the thesis a software was made, which enables use of nine different algorithms, which are 5 k-means clustering algorithms and 4 methods for choosing initial centers. Using real life and synthetic datasets an overview of the application's capabilities is given by measuring algorithms performance, memory use and approximation capabilities.

Keywords:

k-means, cluster analysis, algorithms

CERCS: P175 Informatics, systems theory

Sisukord

1. Sissejuhatus

Üks, võimalus et muuta andmed andmekaeves arusaadavamaks, on läbi viia klasteranalüüs. Klasteranalüüs grupeerib andmeid tunnuste järgi, leides objektid on üksteisele sarnasemad. Tegemist on aktuaalse ülesandega, millele viitab näiteks ka selle lahendamiseks olevate erinevate meetodite paljusus. Kuna erinevad meetodid ei pruugi anda sama või soovitud tulemust, tuleb välja valida üks või rohkem meetodit, mida rakendada. Käesolevas bakalaureusetöös keskendutakse k-keskmiste meetodile, mille eesmärgiks on minimeerida ruutvigade summat klastritesse kuuluvate andmeobjektide ja nende keskpunktide vahel [1].

Bakalaureusetöö käigus valmis autori poolt rakendus programmeerimiskeeles C, mida on võimalik kasutada k-keskmiste optimeerimisülesandel põhineva klasterduse läbiviimiseks. Samuti võiks edaspidi antud rakendust kasutada ka uute algoritmide väljatöötamisel, kuna on lihtne võrrelda tulemusi juba programmis implementeeritud algoritmideg. Rakendus sisaldab endas viite k-keskmiste meetodile lähendit otsivat algoritmi ja nelja algsete keskpunktide valimise meetodit. Andmeobjektide omavahelise kauguse määramiseks on võimalik kasutada kahte erinevat meetrikat. Kasutades loodud rakendust võrreldakse algoritmide implementatsioonide täitmisaega, mälukasutust ja edukust leida head lähendit k-keskmiste ülesandele.

Töö esimeses osa alguses kirjeldatakse antud teema valimise põhjuseid ja eesmärki, mida üritati saavutada. Järgneb ülevaade k-keskmiste meetodist kui optimeerimisülesandest ja vajalikust taustainfost, kus antakse implementeeritud k-keskmiste algoritmide teoreetilised tööpõhimõtted kohta ja pseudokoodid, millest algoritmide implementeerimisel lähtuti. Esimese osa lõpus antakse ülevaade praegustest olemasolevatest rakendustest, mis võimaldavad kasutada k-keskmiste algoritme klasteranalüüsi sooritamiseks.

Teine osa keskendub töö teostaja poolt loodud rakendusele. Esiteks tuuakse välja, mille poolest erineb autori loodud rakendus teistest olemasolevatest rakendustest. Antakse ülevaade autori poolt tehtud tehnilistest valikutest. Järgnevalt antakse programmi kasutamiseks vajalikke baasteadmised: kuidas installeerida loodud programmi ning millised on sisend- ja väljundfailide formaadid. Samuti tuuakse üks näide tarkvara kasutamisest. Tuuakse välja ka rakenduse piirangud ning demonstreeritakse, kuidas rakendus leiab teise olemasoleva rakendusega sama lähendi.

Töö kolmandas osas võrreldakse rakenduses implementeeritud k-keskmiste algoritmide täitmisaega, mälukasutust ja k-keskmiste ülesandele lähendi otsimise võimekust, kasutades selleks kahte kahte sünteetilist ka kahte reaalse elu andmetel põhinevat andmestikku. Kokkuvõttes arutletakse võimalikest loodud programmi edasiarendustest ning antakse lühiülevaade saavutatud tulemustest.

Lisadena on esiteks kaasas *zip* failina on loodud programmi lähtekood, lähtekoodi dokumentatsioon, rakenduse töö näitlikustamiseks mõeldud andmestik ja skriptid visualiseerimiseks ning rakenduse testimiseks. Töö lõpus on lisadena terminoloogia ja litsents töö avaldamiseks.

2. Töö eesmärk

Töö eesmärgiks seati autori poolt luua tarkvara, mis võimaldaks rakendada mitmeid erinevaid k-keskmiste klasteranalüüsiks kasutatavaid algoritme, mis oleksid efektiivsemad kui mitmetes teistes rakendustes kasutatavad k-keskmiste algoritmid. Antud projekti teema püstitamise peamisteks ajenditeks oli koostaja huvi klasteranalüüsi ja andmekaeve vastu.

Uurides olemasolevaid rakendusi leidis bakalureusetöö autor, et tihtipeale ei kasutata neis piisavalt efektiivseid k-keskmiste algoritme, vaid piirdutakse standardse Lloydi algoritmiga [2]. Töö autor pani tähele, et enamik tarkvarasid, mis võimaldavad kasutada mitmeid erinevaid k-keskmiste algoritme ning see juures pakuvad ka mitmeid viise algsete keskpunktide valimiseks, eeldavad suurte raamisitike installeerimist.

Sellega seadis bakalaureusetöö koostaja enda töö eesmärgiks luua uus tarkvara, mis võimaldab klasterdada andmeobjekte kasutades mitmeid erinevaid k-keskmiste algoritme ning ka antud algoritmide jaoks vajalikke algsete keskpunktide valimise algoritme. Seejuures lubaks ka kasutada mitut erinevat meetrikat. Loodud tarkvara peaks olema kergesti kasutatav, kasutama intensiivsete arvutuste jaoks sobilikku programmeerimiskeelt ja ei omaks sõltuvusi teistest rakendustest. Samal ajal peaks rakendus olema ka hästi dokumenteeritud, et soovi korral oleks arendajatel selle edasiarendamine lihtsam.

Implementeeritavate algoritmide võimekuse hindamiseks seati töö osaks ka nende efektiivsuse hindamine.

3. Taustainfo

Käesolevas peatükis antakse rakenduses implementeeritud algoritmide teoreetiline taust. Kõigepealt tutvustatakse k-keskmiste meetodit ning seejärel kirjeldatakse kõiki implementeeritud algoritme ükshaaval.

Andmeobjektide ja klastrite keskpunktide all mõeldakse käesolevas töös d-mõõtmelisi vektoreid, kus vektorite elementideks on reaalarvud $(a_1, a_2, \dots, a_d)$.

3.1 Optimeerimisülesanne k-keskmised

Sõna "*k-means*" ehk k-keskmised kasutati esimest korda James MacQueeni poolt aastal 1967 [1]. Kui varasemalt kasutati sõna k-keskmised konkreetse algoritmi kohta, siis tänapäeval kasutatakse arvutiteaduste valdkonnas seda sõna pigem üldise optimeerimisülesande kohta, millele k-keskmised kui algoritm lähendit pakkus. Alguses k-keskmiste all tuntud algoritmi nimetatakse nüüd pigem Lloydi algoritmiks [2].
Optimeerimisülesandeks, mida k-keskmiste algoritmid lahendada üritavad, on jagada n andmeobjekti k klastrisse nii, et oleks minimeeritud summade summa J (vt joonis 1)

$$J = \sum_{j=1}^{k} \sum_{i=1}^{N(j)} \left| x_{ji} - c_j \right|^2,$$

kus c_j on j-nda klastri keskpunkt, N(j) on andmeobjektide arv j-ndas klastris, x_{ji} on j-nda klastri i-ndas andmeobjekt ja $\left| x_{ji} - c_j \right|$ on kaugusfunktsioon arvutamaks kaugust andmeobjekti x_{ij} ja klastri keskpunkti c_j vahel. Antud summade summat kutsutakse ruutvea funktsiooniks ning optimeeritavaks kriteeriumiks on seega ruutviga.

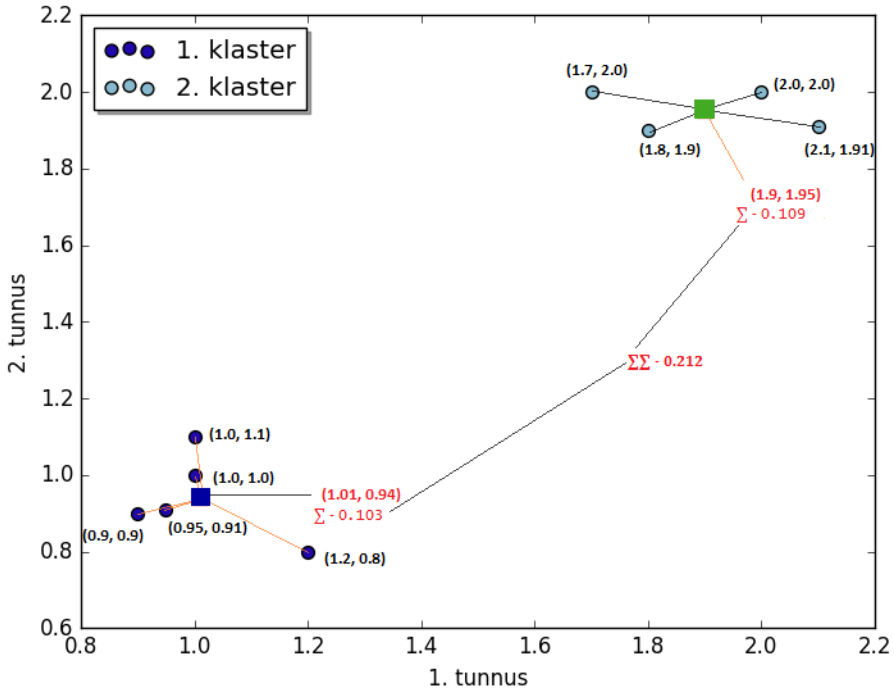
Peamisteks põhjusteks, miks k-keskmiste meetodit kasutatakse, on algse standardse algoritmi ning klasterdamisel ruutvea minimeerimise lihtsus [2].

Miinusteks on antud meetodi puhul näiteks, et klastrite arvu k peab ette andma. Klastrite arvu k leidmiseks kasutatakse erinevaid meetodeid[1], mida käesolevas töös ei käsitleta. Kuna k-keskmiste meetod minimeerib ruutvigade summat, siis ei sobi antud meetodit kasutada igasuguste sisemiste struktuuridega andmestikel. Näiteks ei anna k-keskmiste meetod spiraalse sisemise strukutuuriga andmestikul kvaliteetset klasterdust (vt joonis 2), kuid hea lahenduse võiks leida näiteks tiheduspõhine klasterdusmeetod DBSCAN [3] (vt joonis 3). Samuti sõltub k-keskmiste meetodite poolt antav tulemus suuresti algkeskpunktide valikust [4].

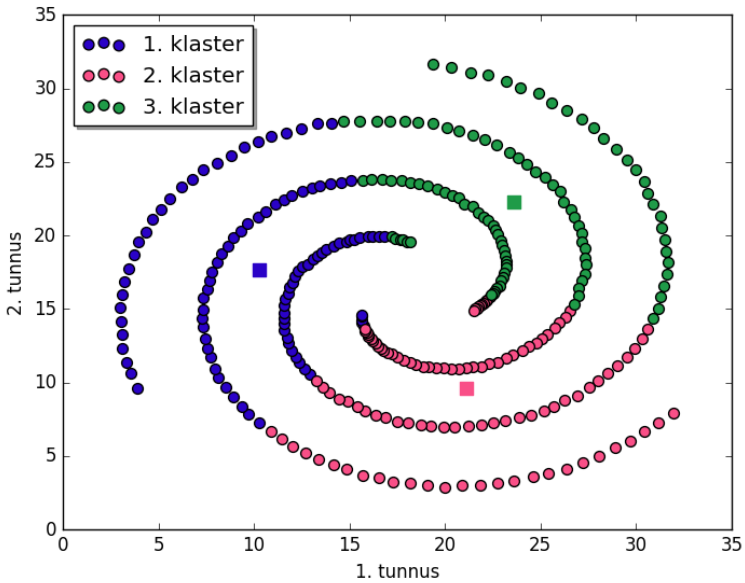
Keerukus

Optimeerimisülesandena on k-keskmiste täpse lahendi leidmine NP-täielik [5], mis tähendab, et praktiliseks kasutamiseks on vaja lähendeid otsivaid algoritme. Kui klastrite k ja dimensioonide arv d on fikseeritud, siis täpse lahenduse ajaline keerukus k-keskmiste ülesandele on $O(n^{dk+1} * \log(n))$, kus n on andmeobjektide arv [6].

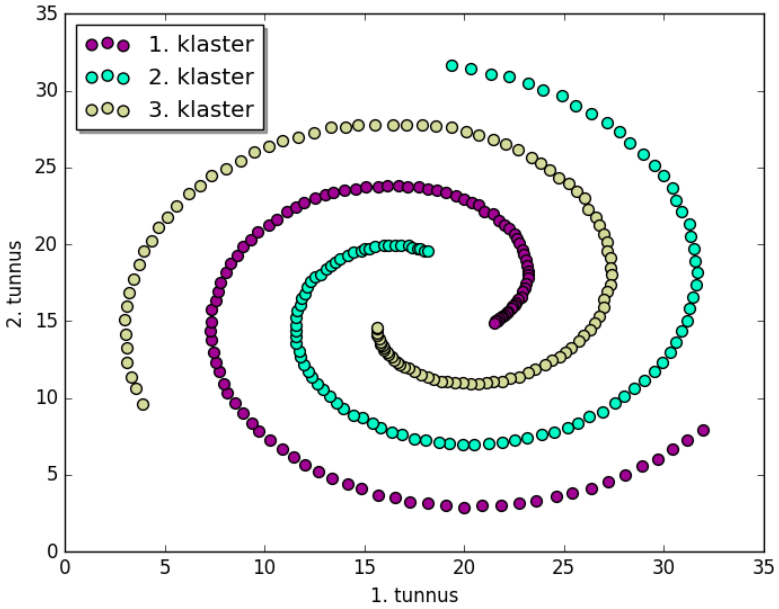
[1] http://www.statsoft.com/Textbook/Cluster-Analysis#vfold (viimati vaadatud 12.05.2016)



Joonis 1. Eukleidilises ruumis ruutvigade summa leidmine. Ruuduga on tähistatud klastri keskpunkti sirglõiguga ühendatud objektidele. ∑-ga tähistatud arv tähendab ühe klastri ruutvigade summat. ∑∑-ga tähistatud arv tähistab ruutvigade summat üle mõlema klastri. 1. klastri ruutvigade summa on 0,103, 2. klastril 0,109. Kokku on klastrisiseste ruutvigade summa 0,212.



Joonis 2. Klastrisiseste ruutvigade summade minimeerimine ei pruugi anda spiraalselt struktureeritud objektidel soovitud tulemust. Ruut tähistab keskpunkti sama värvi klastrile. Visuaali loomiseks on kasutatud andmestikku *Spiral* [7].



Joonis 3. Soovitud tulemus spiraalselt struktureeritud objektidel, mida suudaks leida näiteks tiheduspõhine klasterdusalgoritm DBSCAN [3]. Visuaali loomiseks on andmestikku *Spiral* [7].

3.2 Kaugusmõõt

Optimeerimisülesandena hinnatakse k-keskmiste puhul objektide sarnasust kasutades kauguseid, mistõttu on vaja defineerida kahe objekti vahelise kauguse mõõtmine.
Järgnevalt kirjeldatakse kahte meetrikat, mida loodud tarkvara lubab rakendada. Nii eukleidilise[2] kui ka Manhattani[3] kauguse puhul on tegemist meetrikaga. Meetrika üks omadus eeldab, et suvalise kolme punkti x, y ja z puhul kehtib kolmnurga võrratuse omadus[4] $d(x,z) \leq d(x,y) + d(y,z)$.

Eukleidiline kaugus

Esimeseks kasutatavaks [8] kaugusemõõduks valiti eukleidiline kaugus, kuna tegemist on ühe sagedamini kasutatava kaugusmõõduga. Eukleidiline [8] kaugus d(x, y) arvutatakse kahe r-mõõtmelise reaalarvu vektori $x = (x_1, x_2, .., x_r)$ ja $y = (y_1, y_2, .., y_r)$ jaoks järgnevalt:

$$d(x,y) = d(y,x) = \sqrt{\sum_{i=1}^{r}(x_i - y_i)^2}.$$

Manhattani kaugus

Teise meetrikana kasutatakse rakenduses Manhattani kaugust. Manhattani [8] kaugus kahe r-mõõtmelise vektori $x = (x_1, x_2, .., x_r)$ ja $y = (y_1, y_2, .., y_r)$ jaoks arvutatakse järgnevalt:

$$d(x,y) = d(y,x) = \sum_{i=1}^{r} |x_i - y_i|.$$

[2] http://mathworld.wolfram.com/EuclideanMetric.html (viimati vaadatud 12.05.2016)
[3] http://mathworld.wolfram.com/TaxicabMetric.html (viimati vaadatud 12.05.2016)
[4] http://mathworld.wolfram.com/Metric.html (viimati vaadatud 12.05.2016)

3.3 Algoritmide ülevaade

Antud bakalaureusetöös uuritakse lähemalt üheksat erinevat algoritmi. Nelja algoritmi kasutatakse neist algsete keskpunktide valimiseks ja viit k-keskmiste klasterdamise läbiviimiseks. Kõik algsete keskpunktide valimiseks mõeldud meetodid on mittedeterministlikud, mistõttu võivad nad anda erinevate käitamiskordade ajal erinevaid tulemusi. Tavaliselt viiakse k-keskmiste klasterdust läbi mitu korda ja siis valitakse tulemuste seast välja parim.

Kui standardsel k-keskmiste klasterdusalgoritmil oli algsete keskpunktide valimine kirjeldatud algoritmi osana [2], siis nüüd on rohkem tavaks neid vaadata kui kahte eraldi osa. See tähendab, et siinsed väljatoodud klasterderdusalgoritmid on kõik deterministlikud, kuna mittedeterministlikkus seisneb ainult algsete keskpunktide valikus.

Pseudokoodides mõeldakse koondumise all, et ühe iteratsiooni vältel ei vaheta ükski andmeobjekt oma klastrit ehk keskpunktid ei muutu. Pseudokoodides esinevad peamised tähistused on ära toodud tabelis 1.

Klastri keskpunkti ehk keskmist leitakse klastris olevate andmeobjektide tunnuseid liites ning seejärel iga tunnuse summa jagatakse klastris olevate andmeobjektide arvuga. Kuna andmeobjekti puhul on tegemist reaalarvuliste vektoritega, siis toimib ka liitmine ja jagamine samamoodi nagu eukleidiliste vektorite puhul. Näiteks kui klastris on 3 kolme tunnusega andmeobjekti (0.3, 0.2, 0.5), (1.2, 0.1, 0.5), (0.4, 0.1, 0.7), siis keskpunkt antud klastrile on

$$\frac{(0.3,0.2,0.5)+(1.2,0.1,0.5),+(0.4,0.1,0.7)}{3} = \left(\frac{0.3+0.2+0.5}{3}, \frac{1.2+0.1+0.5}{3}, \frac{0.4+0.1+0.7}{3}\right) = (0.33, 0.6, 0.4).$$

Üldkujul näeb arvutus välja järgnevalt: kui x_1, x_2, ..., x_n, $n \in \mathbb{N}$ on ühte klastrisse kuuluvad n d-dimensionaalset andmeobjekti ja x_{ij} on i-nda andmeobjekti j-s tunnus, siis antud klastri keskpunktiks on $\frac{x_1 + x_2 + \ldots + x_n}{n} = (\frac{x_{11}+x_{12}+\ldots+x_{1d}}{n}, \frac{x_{21}+x_{22}+\ldots+x_{2d}}{n}, \ldots, \frac{x_{n1} + x_{n2} + \ldots + x_{nd}}{n})$

Tabel 1. Peamised pseudokoodides kasutatud tähised

Tähis	Kirjeldus
X	Andmeobjektide hulk.
C	Klastrite keskpunktide hulk.
n	Andmeobjektide arv.
k	Keskpunktide/klastrite arv.
$x(i) \in X, i = \{1, 2, ..., n\}$	i-s andmeobjekt andmeobjektide hulgas.
$c(j) \in C, j = \{1, 2, ..., k\}$	j-nda klastri keskpunkt.
a(i)	Klastri indeks, kuhu i-ndas objekt kuulub.
d(x(i),c(j))	Kaugus i-nda andmeobjekti x(i) ja j-nda keskpunkti c(j) vahel.
d(c(j),c(jj))	Kaugus kahe keskpunkti c(j) ja c(jj) vahel.

3.4 Lloydi algoritm

Lloydi algoritm koosneb kahest etapist, mida teostatakse kordamööda kuni tulemuse koondumiseni ehk ükski andmeobjekti ei vaheta enam klastrit, kuhu ta kuulub [2]:

1) Iga andmeobjekti jaoks arvutatakse kaugus iga klastri keskpunktini ja määratakse lähima kaugusega keskpunkti klastrisse (vt joonis 1, read 3-7).
2) Iga klastri jaoks arvutatakse temasse kuuluvate andmeobjektide keskmine, millest saab vastava klastri uus keskpunkt (vt joonis 1, read 8-9).

```
lloyd(X, C):
    korda kuni keskpunktid pole koondunud:
        iga i ∈ {1, 2, ..., n} korral: {Leiab lähima keskpunkti iga x(i) jaoks}
            a(i) = 1;
5           iga j ∈ {1, 2, ..., k} korral:
                kui d(x(i), c(j)) < d(x(i), c(a(i))) siis:
                    a(i) = j;
        iga j ∈ {1, 2, ..., k} korral: {Liigutab keskpunktid klastrite keskele}
            c(j) = arvuta uus c(j) keskmine;
```

Joonis 4. Lloydi algorimi pseudokood. Kasutatud tähiste kirjeldused on tabelis 1.

3.5 Elkani algoritm

Elkani algoritm [9] on Lloydi algoritmi edasiarendus, mis kasutab meetrikate kolmnurga võrratuse omadust, et jätta vahele andmeobjektide ja keskpunktide vahelisi kauguste arvutusi. Elkani algoritm annab alati sama klasterduse nagu Lloydi algoritm [9]. Algoritm põhineb vaatlusel, et hilisemates iteratsioonides keskpunktid enam eriti ei muutu ja valdav osa andmeobjektidest enam oma klastrit ei vaheta [9]. Kolmnurga võrratuse rakendamiseks jätab Elkani algoritm üle iteratsioonide meelde teatud andmeobjektide ja keskpunktide vahel ülemisi ja alumisi kauguste tõkkeid, mida korrigeeritakase vastavalt keskpunktide väärtuste muutumistele [9]. Algoritmi töö lõpptingimuseks on, et iteratsiooni jooksul ei muutu ükski klaster.

Tabel 2. Elkani algoritmi pseudokoodis kasutatud lisatähised

Tähis	Kirjeldus
cc(c, c')	Keskpunkti c ja c' vaheline kaugus.
u(i)	Ülemine tõke i-nda andmeobjekti jaoks. Tõkestab kauguse andmeobjekti ja ta lähima keskpunkti vahel.
l(i, j)	Alumine tõke i-nda andmeobjekti ja j-nda keskpunkti jaoks.
m(j)	j-nda keskpunkti kaugus oma eelmise iteratsiooni versioonist.
r(i)	Tõene parajasti siis kui iteratsiooni vältel on i-nda andmeobjekti jaoks tõkkeid kitsendatud.
s(j)	Pool j-nda keskpunkti kaugust temale lähimasse teise keskpunkti.

Elkani algoritmi tööpõhimõte keskendub põhiliselt kahele lemmale, mis on tõestatud tuginedes kolmnurga võrratusele [9]:

1) Lemma 1: Olgu x andmeobjekt, b ja c keskpunktid. Kui $d(b,c) \geq 2d(x,b)$, siis $d(x,c) \geq d(x,b)$.
2) Lemma 2: Olgu x andmeobjekt, b ja c keskpunktid. Siis kehtib $d(x,c) \geq \max(0, d(x,b) - d(b,c))$.

Antud lemmadele toetudes saadi järgmised tulemused, mis lubavad kauguste arvutusi vahele jätta [9]:

1) Kui ei teata täpselt d(x,c), aga teatakse ülemist tõket u nii, et $u \geq d(x,c)$. Siis peab arvutama d(x,c') ja d(x,c) ainult siis, kui $u > \frac{1}{2}d(c,c')$. Vastasel juhul saab kindlalt väita, et lähimaks keskpunktiks on c.
2) Kui $u(x) \leq \frac{1}{2}\min \mathrm{d}(c,c')$ üle kõikide c ≠ c', siis andmeobjekt x jääb keskpunkti c juurde.
3) Kui eelmisest iteratsioonist on teada alumine tõke l', et kehtib $d(x,b') \geq l'$, siis me saame arvutada alumise tõkke praegusele iteratsioonile: $d(x,b) \geq \max(0, d(x,b') - d(b,b)') = l$.
4) Kui $u(x) \geq d(x,c)$ on ülemine tõke andmeobjekti x ja tema klastri keskpunkti vahelisele kaugusele ja $l(x,c') \leq d(x,c')$ on alumine tõke andmeobjekti x ja mõne teise keskpunkti c' jaoks ja kui $u(x) \leq l(x,c')$ siis $d(x,c) \leq u(x) \leq l(x,c') \leq d(x,c)$, mis tähendab, et d(x,c) ja d(x,c') arvutused võib vahele jätta.
5) Andmeobjekt x määratakse oma praeguse keskpunkti c juurest keskpunkti c' klastrisse, kui tõkked ei luba arvutust vahele jätta ja leitakse, et $d(x,c') < d(x,c)$.

Algoritmi töö alguses algväärtustatakse tõkked [9]. Ülemised tõkked piiravad andmeobjekti ja lähima keskpunkti võimalikku maksimaalset kaugust. Alumised tõkked piiravad andmeobjekti ja keskpunkti jaoks minimaalset võimalikku kaugust. Algselt algväärtustatud tõkked on sellised, mis kukuvad igaljuhul kontrollidest läbi ja sunnitakse algoritmi arvutama välja täpsed tõkked (vt joonis 5, read 2-6). Peamise tsükli iga iteratsiooni alguses arvutatakse paarikaupa kaugused kõigi keskpunktide vahel ja leitakse iga keskpunkti kaugus jaoks temale lähima teise keskpunkti jaoks (vt joonis 5, read 6-10). Järgnevalt itereeritakse üle kõigi andmeobjektide ja kontrollitakse 2. tulemuse kehtivust ja jäetakse andmeobjekti jaoks arvutused vahele, kui seda käesolev tulemus lubab (vt joonis 5, rida 14). Kui arvutusi ei saa vahele jätta tulemus 2 põhjal, siis hakatakse võrdlema praeguse andmeobjekti keskpunkti ja teiste keskpunktide vahelisi kauguseid esimese ja neljanda tulemuse tingimuse kontrolliks (vt joonis 5, read 16-18). Juhul, kui ka eelnev ei luba arvutusi vahele jätta ja seda ei luba ka uuesti arvutatud ülemine tõke (vt joonis 5, read 19-22), siis kontrollitakse 5. tulemuse kehtivust ja määratakse andmeobjekt teise klastrisse, kui seda tulemus nõuab.

Tõkked [9] arvutatakse uuesti siis, kui nad ei luba vahele jätta kauguste arvutust ja kui neid pole iteratsiooni jooksul juba uuesti arvutatud (vt joonis 5, read 20 ja 23) ehk siis iga kord kui leitakse täpne kaugus andmeobjekti ja keskpunkti vahel kitsendatakse tõkkeid. Tõkete uuesti arvutamist nimetatakse kitsendamiseks.

Iga iteratsiooni [9] lõpus toimub kõigepealt analoogselt uute keskpunktide arvutamine ning siis tõkete lõdvendamine kolmanda tulemuse põhjal: kui c(j) on j-nda klastri keskpunkt ja

c(j)’ on sama klastri keskpunkt eelmises iteratsioonist, siis igale ülemisele tõkkele u(j) liidetakse d(c(j), c(j)’) ja igast alumisest tõkkest lahutatakse d(c(j), c(j)’) (vt joonis 5, read 29-33).

Kui n on klasterdavate andmeobjektide arv ja k on keskpunktide arv, siis Elkani algoritmi kasutatavuse probleemiks võib saada vajatav lisamälu n∗k alumise tõkke ja k∗k klastritevaheliste kauguste jaoks [9].

```
elkan(X, C):
    iga i ∈ {1, 2, ..., n} korral: {sätitakse paika algsed ülemised ja alumised tõkked}
        a(i) = 1;
        u(i) = ∞;
5       iga j ∈ {1, 2, ..., k} korral:
            l(i, j) = 0;
    korda kuni keskpunktid pole koondunud:
        iga j ∈ {1, 2, ..., k} korral: {arvutatakse kaugused kõigi keskpunktide vahel}
            iga jj ∈ {1, 2, ..., k} korral:
10              cc(j, jj) = d(c(j), c(jj));
        iga j ∈ {1, 2, ..., k} korral: {leitakse igale keskpunktile kaugus teise lähimasse keskpunkti}
            s(j) = min j ≠ jj (cc(j, jj) / 2);
        iga i ∈ {1, 2, ..., n} korral:
            kui u(i) ≤ s(a(i)): siis jätka järgmise i-ga; {2. tulemuse kehtivuse kontroll}
15          r(i) = true;
            iga j ∈ {1, 2, ..., k} korral:
                z = max(l(i ,j) , cc(a(i), j) / 2);
                kui j == a(i) või u(i) ≤ z: siis jätka järgmise j-ga; {1 ja 4 tulemuse kontroll}
                kui r(i) siis: {ülemise tõkke kitsendamine}
20                  u(i) = d(x(i), c(a(i)));
                    r(i) = false;
                kui u(i) ≤ z: siis jätka järgmise j-ga;
                l(i, j) = d(x(i), c(j));
                kui l(i, j) < u(i) siis: a(i) = j; {5. tulemus}
25      iga j ∈ {1, 2, ..., k} korral: {keskpunktide liigutamine ja m(j) arvutamine}
            c_koopia = c(j);
            c(j) = arvuta uus c(j) keskmine;
            m(j) = d(c_koopia, c(j));
        iga i ∈ {1, 2, ..., n} korral: {ülemiste tõkete lõdvendamine}
30          u(i) = u(i) + m(a(i));
            iga j ∈ {1, 2, ..., k} korral: {alumiste tõkete lõdvendamine}
                l(i, j) = l(i, j) – m(j);
```

Joonis 5. Elkani algoritmi pseudokood [9, 10]. Üldised kasutatud tähiste kirjeldused on tabelis 1 ja Elkani algoritmi lisatähistused on tabelis 2.

3.6 Hamerly algoritm

Hamerly algoritm [11] on Elkani algoritmi variatsioon, mis kasutab samuti arvutuste vahele jätmiseks ära kolmnurga võrratuse omadust. Nagu Elkani algoritmgi annab Hamerly algoritm sama klasterduse, mille annab Lloydi algoritm [11]. Võrreldes Elkani algoritmiga on Hamerly algoritm lihtsamini implementeeritav ja kasutab vähem mälu, aga jätab selle eest vähem arvutusi vahele [11]. Algoritmi lõpptingimuseks on, et ühe iteratsiooni jooksul ei muutu ükski klaster.

Tabel 3. Hamerly algoritmi pseudokoodis kasutatud lisatähised

Tähis	Kirjeldus

u(i)	Ülemine tõke i-nda andmeobjekti jaoks. Tõkestab kauguse andmeobjekti ja ta lähima keskpunkti vahel.
l(i)	Alumine tõke i-nda andmeobjekti jaoks. Tõkestab andmeobjekti minimaalset kaugust suvalise teise keskpunktiga, mille klastrisse ta hetkel määratud pole.
s(j)	j-nda keskpunkti kaugus oma eelmise iteratsiooni versioonist.
m(j)	j-nda keskpunkti liikumine iteratsiooni jooksul.

Hamerly [11] algoritmi ülemine tõke u(i) on täpselt sama nagu Elkani algoritmi ülemine tõke, mis piirab kaugust andmeobjekti ja tema lähima keskpunkti vahel. Kui Elkani algoritm kasutas iga andmeobjekti jaoks klastrite arvu k jagu alumist tõket, siis Hamerly algoritm kasutab ainult ühte l(i), mis piirab andmeobjekti minimaalset kaugust suvalise teise keskpunkti vahel, millega ta hetkel seotud pole. Samuti ei ole Hamerly algoritmi puhul vaja meelde jätta paarikaupa kauguseid kõigi keskpunktide vahel.

Algoritmi töö algab analoogselt Elkani algoritmiga [11], kus kõigepealt määratakse sellised tõkked, mis arvutatakse kindlalt uuesti (vt joonis 6, read 2-5). Järgnevalt leitakse iga keskpunkti jaoks poolkaugus temale lähimasse teise keskpunkti (vt joonis 6, read 7-8). Seejärel itereeritakse üle kõigi andmeobjektide. Kui andmeobjekti ülemine tõke, mis piirab maksimaalset kaugust andmeobjekti ja ta praeguse klastri keskpunkti vahel on väiksem või võrdne alumise tõkkega, mis piirab andmeobjekti minimaalset kaugust mõne teise keskpunktiga, siis saab kindlalt väita, et ükski teine keskpunkt ei saa antud andmeobjektile lähemal olla kui praegune keskpunkt (vt joonis 6, read 10-11).

Kui aga alumine tõke on väiksem ülemisest tõkkest, siis võib mingi teine keskpunkt olla lähemal kui praeguse klastri keskpunkt [11]. Teada saamiseks, kas mingi teine keskpunkt on lähemal, arvutatakse kõigepealt uuesti välja täpne ülemine tõke praeguse andmeobjekti ja tema keskpunkti vahel (vt joonis 6, rida 12). Võib juhtuda, et nüüd on ülemine tõke väiksem alumisest tõkkest ja võib arvutused vahele jätta (vt joonis 6, rida 13). Kui aga alumine tõke on siiski väiksem, siis peab arvutama täpseid kauguseid praegusest andmeobjektist kõikidesse teiste klastrite keskpunktidesse ja leitakse lähim ja läheduselt teine keskpunkt (vt joonis 6, read 14-15). Kui leitud keskpunkt ei ole sama, mis praegune andmeobjekti klaster, siis määratakse ta sinna ja pannakse paika uus ülemine tõke (vt joonis 6, read 16-18). Kauguselt teist keskpunkti kasutatakse selleks, et määrata alumine tõke (vt joonis 6, rida 19).

Pärast itereerimist [11] üle kõikide andmeobjektide arvutatakse uued keskmised ja jäetakse iga keskpunkti c jaoks meelde kaugust tema eelmisest iteratsiooni versioonist c’ (vt joonis 6, read 20-24). Iteratsiooni lõpus toimub analoogselt Elkani algoritmiga tõkete lõdvendamine (vt joonis 6, read 25-27). Kuna ülemine tõke käitus samamoodi nagu Elkani algoritmi puhul, siis ka siin liidetakse iga andmeobjekti ülemisele tõkkele keskpunkti kaugus eelmise iteratsiooni versioonist. Igast alumisest tõkkest lahutatakse maksimaalne kaugus eelneva iteratsiooni keskpunktide ja vastavate uute keskpunktide vahel.

```
hamerly(X, C):
    iga i ∈ {1, 2, ..., n} korral:
        a(i) = 1;
        u(i) = ∞;
5       l(i) = 0;
    korda kuni keskpunktid pole koondunud:
        iga j ∈ {1, 2, ..., k} korral:
            s(j) = min j ≠ jj d(c(j), c(jj)) / 2
        iga i ∈ {1, 2, ..., n} korral:
10          z  = max(l(i), s(a(i)));
            kui u(i) ≤ z siis: jätka järgmise i-ga;
            u(i) = d(x(i), c(a(i)));
            kui u(i) ≤ z siis: jätka järgmise i-ga;
            c1 = arg min j ∈ {1, 2, .., k} d(x(i), c(j));
15          c2 = arg min j ∈ {1, 2, ..., k} / { c1} d(x(i), c(j));
            kui c1 ≠ a(i) siis:
                a(i) = c1;
                u(i) = d(x(i), c(a(i)));
            l(i) = d(x(i), c(a(c2)));
20      iga j ∈ {1, 2, ..., k} korral:
            c_koopia = c(j);
            c(j) = arvuta uus c(j) keskmine;
            m(j)  = d(c_koopia, c(j));
        mc = max j ∈ {1, 2, ..., k} m(j);
25      iga i ∈ {1, 2, ..., n} korral:
            u(i) = u(i) + m(a(i));
            l(i) = l(i) - mc;
```

Joonis 6. Hamerly algoritmi pseudokood [10, 11]. Üldised kasutatud tähiste kirjeldused on tabelis 1 ja Hamerly algoritmi lisatähistused tabelis 3.

3.7 MacQueeni algoritm

MacQueeni algoritm [1] on üks vanemaid meetodeid k-keskmiste analüüsi läbiviimiseks. Antud algoritmi peamine erinevus Lloydi algoritmist on see, et kui andmeobjektile leitakse uus lähim keskpunkt (vt joonis 7, rida 5), siis seotud klastrite puhul arvutatakse koheselt uued keskpunktid (vt joonis 7, read 10-12). Algoritmi töö lõpptingimuseks on, et ühe iteratsiooni vältel ei muutu ükski klaster.

```
macqueen(X, C):
    iga i ∈ {1, 2, ..., n} korral:
        a(i) = arg min j ∈ {1, 2, ..., k} d(x(i), c(j));
    iga j ∈ {1, 2, ..., k} korral:
5       c(j) = arvuta c(j) keskmine;
    korda kuni keskpunktid pole koondunud:
        iga i ∈ {1, 2, ..., n} korral:
            j = arg min j ∈ {1, 2, .., k} d(x(i), c(j));
            kui j ≠ a(i) siis:
10              a_cp = a(i), a(i) = j;
                c(a_cp) = arvuta uus c(a_cp) keskmine;
                c(j) = arvuta uus c(j) keskmine;
        iga j ∈ {1, 2, ..., k} korral: {Liigutab keskpunktid klastrite keskele}
            c(j) = arvuta uus c(j) keskmine;
```

Joonis 7. MacQueeni algoritmi pseudokood [12]. Kasutatud tähiste kirjeldused on toodud tabelis 1.

3.8 Hartigan-Wongi algoritm

Hartigan-Wongi algoritm [13] on osaliselt sarnane MacQueeni algoritmiga, kuna klastrisse kuuluvust vahetatakse kohe, kui leitakse, et objekti üleviimine teise klastrisse annab optimaalsema tulemuse. Algoritmi töö alguses leitakse iga andmeobjekti x(i) jaoks lähim ja teine lähim klaster, millede indeksid jäetakse meelde vastavalt IC1(i) ja IC2(i).

Tabel 4. Hartigan-Wongi algoritmi pseudokoodis kasutatud lisatähised

Tähis	Kirjeldus
IC1(i)	i-nda andmeobjekti jaoks senileitud optimaalseima klastri indeks.
IC2(i)	i-nda andmeobjekti jaoks senileitud teise optimaalseima klastri indeks.
NC(j)	j-ndasse klastrisse kuuluvate andmeobjektide arv.
R1_mem(i)	i-nda andmeobjekti jaoks meeldejäetud R1.
T1(j)	T1(j) on tõene parajasti siis, kui 2. faasis indeksiga j klaster muutus. 1. iteratsiooni 1. faasis on T1(j) tõene iga i korral.
T2(j)	T2(j) on tõene parajasti siis, kui 1. faasis indeksiga j klaster muutus.
T3(j)	T3(j) on tõene parajasti siis, kui 2. faasi eelmises iteratsioonis indeksiga j klastri kooslus muutus.

Hartigan-Wongi algoritmi [13] puhul ei ole optimeerimise kriteeriumiks andmeobjektide kaugused keskpunktidest, vaid klastrisiseste ruutvigade summade muutused. Andmeobjekt määratakse teise klastrisse, kui see toob kaasa üldise ruutvigade summa vähenemise.

See tähendab, et andmeobjekti x, praeguse klastri L1 ja kandidaadiks oleva klastri L2 puhul võrreldakse järgmiseid [13] arve:

$$R1 = \frac{[NC(L1)*d(x,L1)^2]}{[NC(L1)-1]}, R2 = \frac{[NC(L2)*d(x,L2)^2]}{[NC(L2)+1]}.$$

Kui R2 ≥ R1 ehk kui võtta ära L1 keskpunkti klastrist andmepunkt x ja lisada ta R2 keskpunkti klastrisse, siis üldine ruutvigade summa klasterdusel suureneb suureneb ja optimaalsema tulemuse saab jättes andmepunkt praegusesse klastrisse, vastasel juhul viiakse andmepunkt üle ja arvutatakse uued keskpunktid (vt Joonis 8, read 15-25, 27-36, 47-56).

Hartigan-Wongi algoritmi töö on jaotatud kahte faasi [13], mis omavahel vahelduvad. Esimene faas (vt Joonis 8, read 10-36) on optimaalsete vahetuste faas, kus võrreldakse R1 ja R2 muutumist üle kõikide punktide ja keskpunktide. Kui esimeses faasis muutuseid ei tehtud, siis on jõutud lähendini (vt Joonis 8, rida 37). Teine faas (vt Joonis 8, read 40-57) on kiirete vahetuste faas, kus iga andmeobjekti x(i) jaoks arvutatakse R1 ja R2 ainult parasjagu optimaalseima klastri indeksiga IC1(i) ja teise optimaalsema klastri indeksiga IC2(i) vahel. Teise faasi mõistlikkus tuleneb tähelepanekust, et kui mingisse teise klastrisse üle viies

ruutvigade summa väheneb siis on tihtipeale tegu just teise lähima klastriga. Teist faasi korratakse seni, kuni seal toimub muutusi.

Hartigan-Wongi algoritm teeb lisaks veel mõningaid teisi optimisatsioone [13]. Näiteks kui andmeobjekt kuulub klastrisse, mis pole vahepeal muutunud, siis on vaja arvutada R2 ainult nende klastrite jaoks, mis on vahepeal muutunud (vt joonis 8, read 26-36). Samuti jäetakse iga andmeobjekti jaoks meelde tema R1 väärtus, mis arvutatakse uuesti ainult siis, kui vastava andmeobjekti klaster on muutunud (vt joonis 8, read 17 ja 47). Kui klastri kooslus pole muutunud, siis saab arvutamise vahele jätta ja saab kasutada meeldejäetud R1 väärtust (vt joonis 8, read 29 ja 49).

```
hartiganwong(X, C):
    iga i ∈ {1, 2, ..., n} korral: {otsi iga objekti jaoks 1. ja 2. lähim klaster}
        IC1(i) = arg min j ∈ {1, 2, .., k} d(x(i), c(j));
        IC2(i) = arg min j ∈ {1, 2, ..., k} / { c1} d(x(i), c(j));
5       a(i) = IC1(i); {määrame objekti IC1(i)-ndasse klastrisse}
    iga j ∈ {1, 2, ..., k} korral:
        c(j) = j-ndasse klastrisse kuuluvate andmeobjektide keskmine;
        iga j ∈ {1, 2, ..., k} korral: {kõik klastrid on töötlemishulgas}
            T1(j) = true; {j-ndas klaster on 1. faasi töötlemisnimekirjas}
10  korda:
        iga j ∈ {1, 2, ..., k} korral: {1. faas}
            T2(j) = false;
        iga i ∈ {1, 2, ..., n} korral:
            kui T1(a(i)) siis: {kui a(i) klaster on töötlemisnimekirjas}
15              L2 = arg min j ∈ {1, 2, .., k}/{a(i)} [NC(j)*d(x(i), c(j))]/[NC(j)+1];
                R2 = [NC(L2) * d(x(i), c(L2))] / [NC(L2)+1];
                R1_mem(i) = [NC(IC1(i))*d(x(i), c(IC1(i)))] / [NC(IC1(i))-1];
                kui R2 ≥ R1_mem(i) siis:
                    IC2(i) = L2;
20              vastasel juhul: {R2 < R1_mem(i)}
                    T2(a(i)) = true, T(L2) = true;
                    IC2(i) = IC1(i), IC1(i) = L2;
                    a_cp = a(i), a(i) = L2;
                    c(L2) = arvuta uus c(L2) keskmine;
25                  c(a_cp) = arvuta uus c(a_cp) keskmine;
            vastasel juhul: {T1(a(i)) == false ehk eelmises faasis
                L2 = arg min j ∈ {v | T1(v) = true}/{a(i)} [NC(j) * d(x(i), c(j))]/[NC(j)+1];
                R2 = [NC(L2) * d(x(i), c(L2))] / [NC(L2) + 1];
                kui R2 ≥ R1_mem(i) siis:
30                  IC2(i) = L2;
                vastasel juhul (R2 < R1_mem(i)) siis:
                    T2(a(i)) = true, T2(L2) = true;
                    IC2(i) = IC1(i), IC1(i) = L2;
                    a_cp = a(i), a(i) = L2;
35                  c(L2) = arvuta uus c(L2) keskmine;
                    c(a_cp) = arvuta uus c(a_cp) keskmine;
        kui iga j ∈ {1, 2, ..., k} korral T2(j) == false siis: lõpeta algoritmi töö; {ükski klaster ei muutunud}
        iga j ∈ {1, 2, ..., k} korral: {2. faas}
            T1(j) = false;
40      korda kuni ükski objekt ei vaheta korduse jooksul kuuluvust: {2. faas}
            iga j ∈ {1, 2, ..., k} korral: T3(j) = false;
            iga i ∈ {1, 2, ..., n} korral:
                L1 = IC1(i), L2 = IC2(i);
                kui T2(L1) = false ja T2(L2) = false siis:
45                  jätka järgmise i-ga;
                kui T2(L1) = true siis:
                    R1_mem(i) = [NC(L1)*d(x(i), c(L1))] / [NC(L1)-1];
                    R2 = [NC(L2) * d(x(i), c(L2))] / [NC(L2) + 1];
                kui R2 ≥ R1_mem(i) siis: jätka järgmise i-ga;
```

```
50              vastasel juhul: {R2 < R1_mem(i)}:
                    T1(L1) = true, T1(L2) = true;
                    T3(L1) = true, T3(L2) = true;
                    IC1(i) = L2, IC2(i) = L1;
                    a(i) = L2;
55                  c(L2) = arvuta uus c(L2) keskmine;
                    c(L1) = arvuta uus c(L1) keskmine;
            T2 = T3;
        iga j ∈ {1, 2, ..., k} korral:  T2(j) = false;
```

Joonis 8. Hartigan-Wongi pseudokood [13]. Üldised tähistuste kirjeldused on toodud tabelis 2. Konkreetse algoritmi pseudokoodis kasutatud lisatähistuste seletused on toodud tabelis 4.

3.9 Algsete keskpunktide valimine

Kõik eelnevalt kirjeldatud k-keskmiste algoritmid eeldavad, et on olemas mingi keskpunktide hulk, millest lähtudes hakatakse lähendit otsima. Algsete keskpunktide valimiseks on mitmeid võimalusi. Bakalaureusetöös loodud programmis on implementeeritud neist neli: Forgy meetod, kaugem enne meetod, jagamise meetod ja k-means++ meetod.

Forgy meetod

Forgy meetod on üks lihtsamaid ja levinumaid meetodeid algsete keskpunktide valimiseks. Forgy [14] meetod seisneb selles, et valitakse juhuslikult kõikide andmepunktide seast välja k andmeobjekti, mis saavad algseteks keskpunktideks (vt joonis 9). Forgy meetodi puuduseks on asjaolu, et mitu valitud keskpunkti võivad sattuda üksteise lähedusse (vt joonis 13), mis võib anda kehvema lähendi klasterdusalgoritmi rakendamisel.

```
forgy(X, k):
    korda kuni valitud on k keskpunkti:
        c = vali juhuslikult suvaline andmeobjekt x ∈ X\C
        Lisa c kõikide keskpunktide hulka C
```

Joonis 9. Forgy meetodi pseudokood. Tähistuste kirjeldused on tabelis 1.

Kaugem enne meetod

Furthest-First ehk kaugem enne meetod [15] on algsete keskpunktide valimiseks loodud meetod, mille idee seisneb senivalitud keskpunktidest kaugeima andmeobjekti valimises järgmiseks keskpunktiks. Esimene keskpunkt valitakse andmeobjektide seast juhuslikult ning see on ainus mittedeterministlik samm algoritmis (vt joonis 10, rida 2). Järgnevalt otsitakse iga andmeobjekti jaoks lähim keskpunkt. Andmeobjekt, mille lähim keskpunkt on kõigi seast kaugeim, valitakse järgmiseks keskpunktiks (vt joonis 10, read 4-5). Sama korratakse, kuni on valitud k keskpunkti.

Meetodi tulemusena on valitud keskpunktid üksteisest võimalikult kaugel (vt joonis 14). Probleemiks antud meetodi puhul on, et kuna keskpunktideks valitakse võimalikult kauged andmeobjektid, siis valitakse keskpunktideks tihti võõrväärtuseid, mis võivad vähendada klasterduse kvaliteeti [16].

```
kaugem_enne(X, k):
    c = vali juhuslikult suvaline andmeobjekt x ∈ X
    Lisa c keskpunktide hulka C
    korda kuni valitud on k keskpunkti:
        c = vali andmeobjekt, mille kaugus tema lähimasse keskpunkti on maksimaalne
5       Lisa c kõikide keskpunktide hulka C
```

Joonis 10. Kaugem enne meetodi pseudokood. Tähistuste kirjeldused on tabelis 1.

Jagamise meetod

Random Partition ehk jagamise meetod [15] töö seisneb selles, et iga andmeobjektile määratakse juhuslikult suvaline klaster (vt joonis 11, read 2-3). Igale klastrile arvutatakse keskmine, mis saab antud meetodi poolt tekitatud keskpunktiks (vt joonis 11, rida 4-5). Algoritmi tulemusena satuvad kõik algkeskpunktid tõenäoliselt andmestiku keskele [15] (vt joonis 15), mistõttu ei ole ta eriti soositud meetod k-keskmiste algoritmide puhul algsete keskpunktide valimiseks.

```
jagamine(X, k):
    Iga i = {1,2,...,n} jaoks:
        a(i) = vali juhuslikult üks täisarv lõigust [1, k]
    iga j ∈ {1, 2, ..., k} korral:
5       c(j) = j-ndasse klastrisse kuuluvate andmeobjektide keskmine;
```

Joonis 11. Jagamise meetodi pseudokood. Tähistuste kirjeldused on tabelis 1.

Algsete keskpunktide valimise k-means++ meetod

Üks rakenduste poolt populaarsemini kasutatav algsete keskpunktide valimise algoritm on k-means++ meetod. k-means++ [17] meetod erineb Forgy meetodist sellega, et andmeobjektide keskpunktiks valimise tõenäosus sõltub ruutvõrdeliselt andmeobjektide kaugustest seni valitud keskpunktideni [17]. Mõneti võiks öelda, et tegemist on kaugem enne ja Forgy meetodi hübriidiga (vt joonis 16), kuna eelistatakse just kaugemaid andmeobjekte.

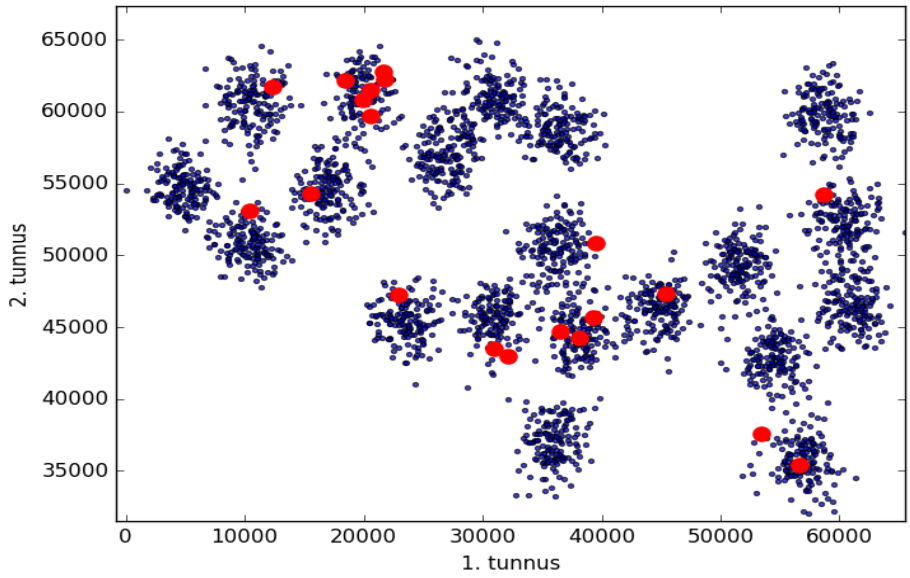
Eelised k-means++ kasutamisel:

1) Keskmiselt leitakse parem lähend k-keskmiste meetodi rakendamisel [17].
2) Algselt valitud paremad keskpunktid vähendavad k-keskmiste meetodi algoritmide tööaega [17].

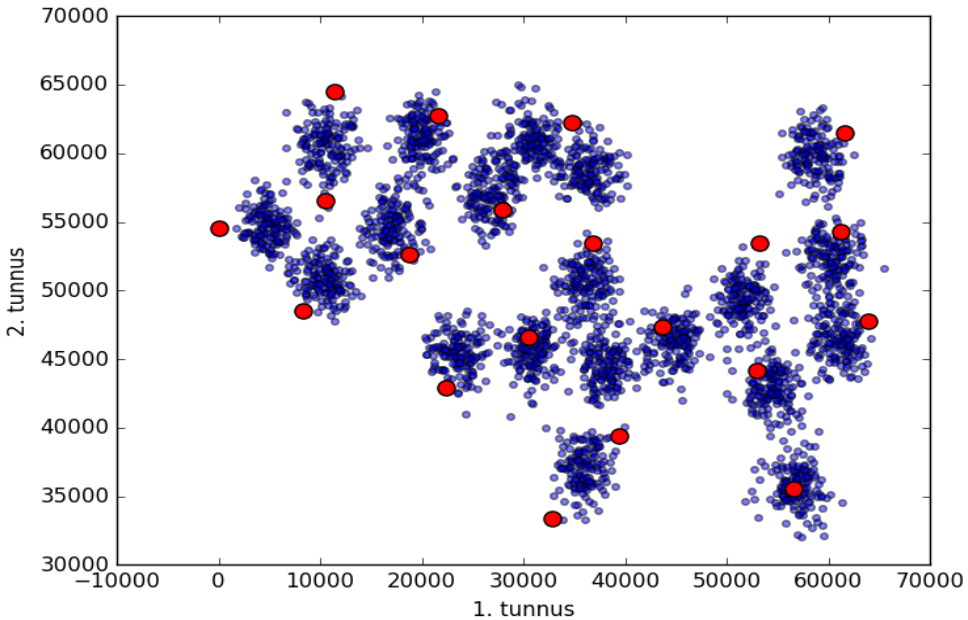
Algoritmi algus on sama nagu Forgy meetodil. Valitakse juhuslikult suvaline andmeobjekt, millest saab esimene keskpunkt (vt joonis 12, rida 2). Järgnevalt leitakse igast andmeobjektist kaugus temale lähimasse keskpunkti. Järgmiseks keskpunktiks valitakse andmeobjekt, kus iga andmeobjekti x_i valimise tõenäosuseks on $min\ c\ \in C \frac{d(x_i,c)^2}{\sum_{p=1}^{n} \mathrm{d}(\mathrm{x_p},\mathrm{c})^2}$ (vt joonis 17), kus C on seni valitud keskpunktide hulk ja n on andmeobjektide arv. Seda korratakse, kuni on valitud k keskpunkti.

Kmeans++(X, k):
    Olgu algselt keskpunktide hulk tühi C = {}
    c = vali juhuslikult suvaline andmeobjekt $x_i \in X$
    Lisa c keskpunktide hulka C
    Korda kuni on valitud k keskpunkti:
        c = vali juhuslikult üks andmeobjekt hulgast $x_i \in X$, objekt x_i valimise tõenäosusks on $min\ c \in C \frac{d(x_i,c)^2}{\sum_{p=1}^{n} d(x_p,c)^2}$
        Lisa c keskpunktide hulka C

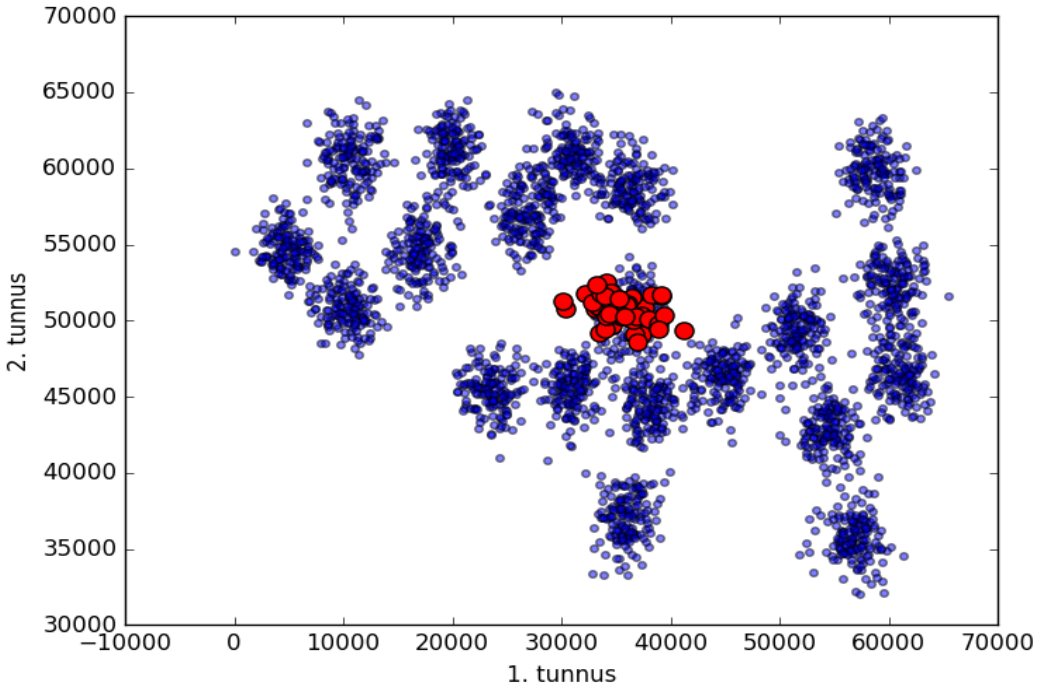
Joonis 12. k-means++ algkeskpunktide valimise algoritmi pseudokood. Tähistuste kirjeldused on tabelis 1.



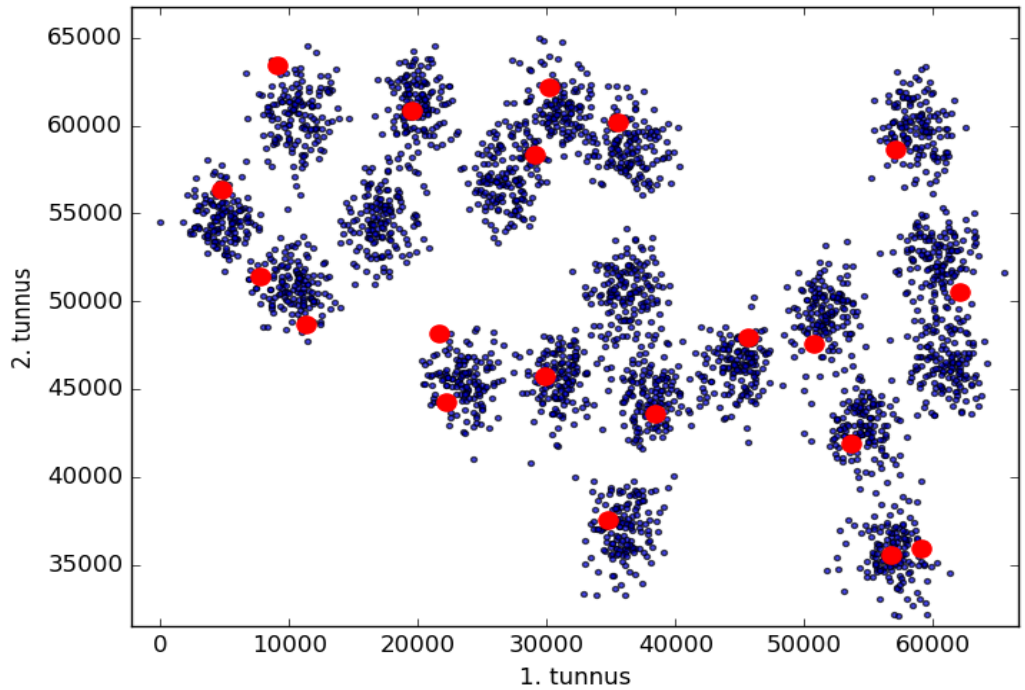
Joonis 13. Forgy meetodi korral on võimalik, et mitu algset keskpunkti satuvad üksteise lähedusse. Forgy meetodi rakendamine A1 [18] andmestiku peal.



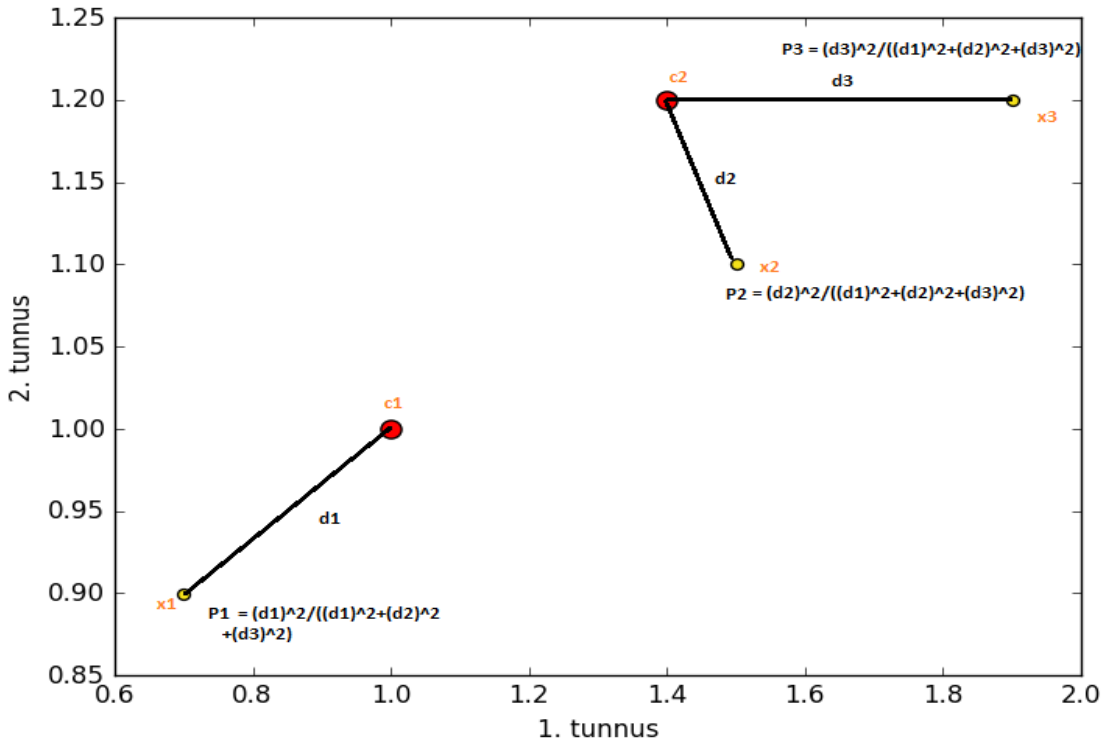
Joonis 14. kaugem enne meetodi rakendame A1 [18] andmestikul. Keskpunktid on võimalikult hajutatud, kuid on tundlik võõrväärtustele.



Joonis 15. jagamise meetodil saadud algsete keskpunktide valik A1 [18] andmestikul. Jagamise meetodi tulemusena on keskpunktid sattunud andmestiku keskele.



Joonis 16. k-means++ jaotab suurema tõenäosusega algsed keskpunktid paremini üle andmestiku laiali, andes seejuures ka tõenäosusliku võimaluse vältida kaugem enne meetodi poolt valitavaid võõrväärtuseid. Joonisel on k-means++ rakendamine A1 [18] andmestiku peal.



Joonis 17. k-means++ meetodi tõenäosused valimaks järgmist keskpunkti, kus c1 ja c2 on juba valitud keskpunktid ja x1, x2 ning x3 on kandidaadid. x1-le on lähim keskpunkt c1 (kaugus d1), x2-le (kaugus d2) ja x3-le (kaugus d3) c2. Seega k-means++ algoritmi kohaselt on tõenäosus järgmiseks keskpunktiks x1 valida P1, x2 valida P2 ja x3 valida P3.

3.10 Olemasolevate rakenduste ülevaade

Rakendusi, mis võimaldavad kasutada k-keskmiste meetodit, on palju. Siin alapeatükis antakse neist osaline ülevaade. Iga rakenduse puhul uuriti, milliseid meetrikaid, k-keskmiste klasterdusalgoritme ja algkeskpunktide valimise meetodeid saab kasutada.

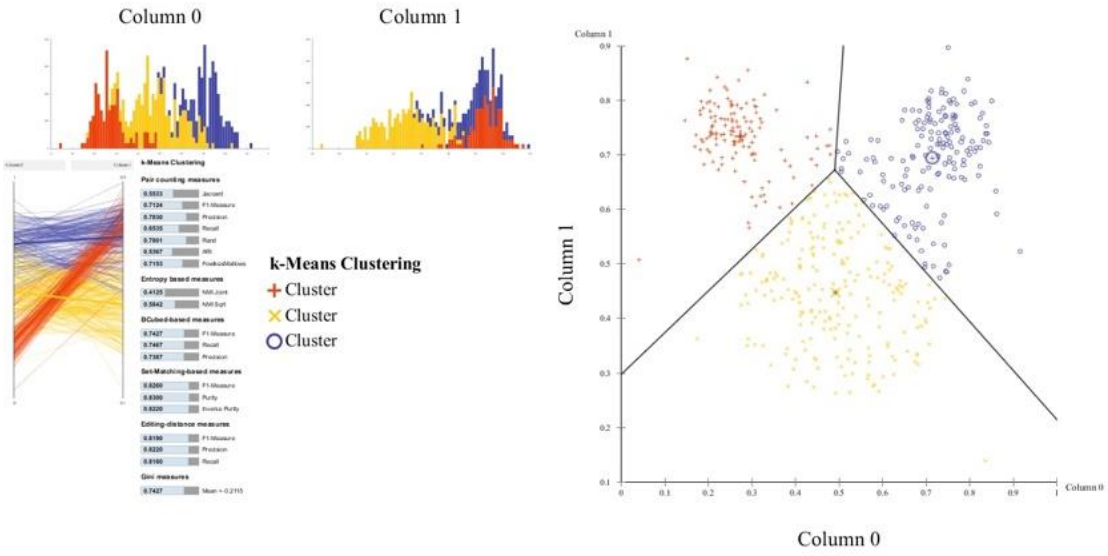
Tarkvara ELKI

ELKI ehk *Environment for Developing KDD-Applications Supported by Index Structures* on vabavaraline Javas kirjutatud andmekaeve rakendus, mille eesmärgiks on olla platvormiks andmekaeve algoritmide arendamisel. ELKI sisaldab endas palju erinevaid andmekaeves kasutatavaid algoritme, sealhulgas, aga mitte ainult, ka mitmeid k-keskmiste meetodi algoritme nagu Lloydi, Hamerly, Elkani ja MacQueeni algoritmid.[5]

Algkeskpunktide valimiseks on seal võimalik kasutada kõiki algkeskpunktide valikuid, mida selles bakalaureusetöös on kirjeldatud ja rohkemgi[6]. Rakendusel on graafiline liides ning sisaldab endas ka vahendeid klasterduste visualiseerimiseks (vt joonis 18).

[5] http://elki.dbs.ifi.lmu.de/releases/release0.7.1/doc/de/lmu/ifi/dbs/elki/algorithm/clustering/ClusteringAlgorithm.html (vaadatud 12.05.2016)

[6] http://elki.dbs.ifi.lmu.de/releases/release0.6.0/doc/de/lmu/ifi/dbs/elki/distance/distancefunction/ (vaadatud 12.05.2016)



Joonis 18. ELKI rakenduse Lloydi implementatsiooni rakendamine rakenduse poolt antud näidisandmestikul.

Tarkvara R

Tarkvara R[7] on samuti vabavaraline rakendus, mida kasutatakse statistiliste arvutuste sooritamiseks. Klasteranalüüsi k-keskmiste meetodi algoritmidest on R-is implementeeritud Lloydi, MacQueeni ning Hartigan-Wongi algoritmid ning algkeskpunktide valimiseks kasutatakse ainult Forgy meetodit[8]. Selleks, et kasutada R-i toel k-keskmiste algoritme, eeldab R kogu suure platvormi allalaadimist. Samuti peab kasutama algoritme läbi R-i, mis võib olla kasutaja jaoks ebamugav.

Tarkvara MATLAB

MATLAB[9] on tasuline arenduskeskkond ja programmeerimiskeel, mis võimaldab kasutajatel kergelt sooritada näiteks maatriksoperatsioone ja erinevate andmete visualiseerimist. Klasteranalüüsi k-keskmiste algoritmidest võimaldab MATLAB kasutada Lloydi ja MacQueeni algoritmi[10], meetrikate hulgast lubab kasutada näiteks eukleidilist, Manhattani ja Hammingi kaugusmõõdustikke.

Tarkvara Scikit-learn ja SciPy

Scikit-learn[11] puhul on tegemist Pythoni masinõppe mooduliga, mis on ehitatud SciPy ja NumPy peale. Scikit-learn tarkvaras on implementeeritud bakalaureusetöö koostaja poolt implementeeritavatest algoritmidest Lloydi algoritm ning rakendab eukleidilist kaugust[12]. SciPy kasutab enda sisemuses ära Lloydi algoritmi[13] ja hulganisiti erinevaid kaugusmõõte[14].

[7] https://www.r-project.org/ (vaadatud 12.05.2016)
[8] https://en.wikibooks.org/wiki/Data_Mining_Algorithms_In_R/Clustering/K-Means (vaadatud 12.05.2016)
[9] http://se.mathworks.com/products/matlab (vaadatud 12.05.2016)
[10] http://se.mathworks.com/help/stats/kmeans.html (vaadatud 12.05.2016)
[11] http://scikit-learn.org/ (vaadatud 12.05.2016)
[12] http://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html (vaadatud 12.05.2016)
[13] https://github.com/scipy/scipy/blob/v0.17.0/scipy/cluster/vq.py#L671-L783 (vaadatud 12.05.2016)
[14] http://docs.scipy.org/doc/scipy/reference/spatial.distance.html (vaadatud 12.05.2016)

Cluster 3.0

Cluster 3.0[15]on vabavaraline klasterdustarkvara, mis on kirjutatud programmeerimiskeeles C. See tarkvara lubab kasutada mitmeid erinevaid klasterdus-meetodeid ning k-keskmiste algoritmidest on seal olemas Lloydi algoritm[16]. Algsete keskpunktide valimiseks kasutatakse Forgy meetodit.

KMlocal

KMlocal on programmeerimiskeeles C++ kirjutatud klasterdustarkvara, mis lubab kasutada

KMlocal[17] on programmeerimiskeeles C++ kirjutatud klasterdustarkvara, mis lubab kasutada nelja k-keskmiste algoritmi. Kasutab ära Lloydi algoritm[18]. Algkeskpunktide valimise meetodina kasutab KMlocal Forgy algoritmi.

Parallel K-Means

Parallel K-Means[19] on programmeerimiskeeles C kirjutatud klasterdustarkvara, mis lubab rakendada Lloydi algoritmi paralleelset implementatsiooni, kasutades selleks näiteks OpenMP[20] või OpenMPI[21] platvorme.

[15] http://bonsai.hgc.jp/~mdehoon/software/cluster/software.htm (vaadatud 12.05.2016)
[16] http://bonsai.hgc.jp/~mdehoon/software/cluster/cluster3.pdf (vaadatud 12.05.2016)
[17] https://www.cs.umd.edu/~mount/Projects/KMeans/ (vaadatud 12.05.2016)
[18] https://www.cs.umd.edu/~mount/Projects/KMeans/kmlocal-doc.pdf (vaadatud 12.05.2016)
[19] http://users.eecs.northwestern.edu/~wkliao/Kmeans/ (vaadatud 12.05.2016)
[20] http://openmp.org/wp/ (vaadatud 12.05.2016)
[21] https://www.open-mpi.org/ (vaadatud 12.05.2016)

4. Rakendus

Käesolevas peatükis antakse ülevaade bakalureusetöö autori poolt loodud k-keskmiste algoritme rakendavast tarkvarast. Peatüki alguses tuuakse välja erinevused seni olemasolevatest tarkvaradest. Kirjeldatakse programmi tehnilisi valikuid, mida autor töö käigus tegi. Antakse juhised, kuidas programmi installeerida ja kasutada. Samuti tuuakse peatüki lõpus välja see, kuidas kontrolliti algoritmide rakendamisel saadud lahenduste korrektsust.

4.1 Erinevused olemasolevatest tarkvaradest

Rakenduse peamine erinevus olemasolevatest tarkvaradest seisneb suurte ja väikeste programmide plusside ja miinuste uurimises. Suuremate rakenduste nagu MATLAB ning SciPy plussiks on, et nad lubavad kasutada mitmeid erinevaid algoritme k-keskmiste ülesande lahendamiseks. Miinuseks on see, et nad nõuavad suurte raamistike installeerimist. Väikeste rakenduste plussideks on nende lihtsus ja kerge installeeritavus. Miinuseks on väheste valikute olemasolu – enamus uuritud tarkvaradest lubasid rakendada ainult ühte k-keskmiste klasterdusalgoritmi, meetrikat või algsete keskpunktide valikuvõimalust. Kuigi ELKI puhul on tegemist suhteliselt väikese rakendusega, mis lubab kasutada palju erinevaid k-keskmiste meetodi algoritme, siis saab tema takistuseks see, et programm on kirjutatud programmeerimiskeeles Java, mis ei ole mõeldud suuremahuliste ja intensiivsete arvutuste tegemiseks.

Loodud rakendus üritab seejuures kokku võtta mõlema poole paremad küljed – pakub mitmeid erinevaid meetodeid algsete keskpunktide valikuks ning mitut erinevat k-keskmiste algorimi. Samas on tegemist kergesti installeerivata ja väikesemahulise programmiga, mille kompileeritud rakenduse failisuurus on umbes 100 KB.

4.2 Tehnilised valikud

Tehnilisi valikuid rakenduse loomisel oli vähe, kuna kirjutatud rakendus valmis puhtalt programmeerimiskeeles C. Programmeerimiskeel C osutus valituks mitmel erineval põhjusel. Esiteks oli tegemist programmeerimiskeelega, millega bakalaureusetöö autor oli varem kokku puutunud. Teiseks on antud keele puhul tegemist populaarse keelega teadusarvutuste valdkonnas[22]. Kuna C on teadusarvutuste valdkonnas populaarne, siis on tänapäeval antud keeles näiteks kerge koodi paralleliseerida, kasutades selleks näiteks OpenACC[23] kompilaatorit.

Rakendus on kirjutatud järgides C11[24] standardit, kuna tegemist on hetkel eelistatuima standardiga, kuid täidab ka kõik C99[25] standardi nõuded.

Sisend- ja väljundfailide formaatide valimise puhul mõeldi eelkõige kasutatavuse ja lihtsuse peale (vt 4.3 peatüki alapeatükki “Sisend ja väljundfailide formaadid”).

4.3 Rakenduse ülevaade

Loodud rakenduses on võimalik kasutada kokku üheksat erinevat algoritmi:

1) Lloydi algoritm.

[22] https://en.wikipedia.org/wiki/Computational_science (vaadatud 12.05.2016)
[23] http://www.openacc.org/ (vaadatud 12.05.2016)
[24] http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1570.pdf (vaadatud 12.05.2016)
[25] http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1256.pdf (vaadatud 12.05.2016)

2) Elkani algoritm.
3) Hamerly algoritm.
4) MacQueeni algoritm.
5) Hartigan-Wongi algortim.
6) Forgy algsete keskpunktide valimise meetod.
7) Kaugem enne algsete keskpunktide valimise meetod.
8) Jagamise meetodil algsete keskpunktide valimise meetod.
9) k-means++ algsete keskpunktide valimise meetod.

Samuti lubab rakendus kasutada kauguste mõõtmiseks kahte erinevat meetrikat: eukleidilist kaugust ja Manhattani kaugust.

Selleks, et algoritmide rakendamine ja arendamine oleks mugavam on võimalik anda programmile parameetrina ette ka pseudojuhuslike arvude generaatori seemne. Kasutades sama seemet, tagastavad algsete keskpunktide valimise meetodid alati sama tulemuse.

Rakenduses on võimalik ette määrata maksimaalsete iteratsioonide arv. Seda kasutatakse siis, kui pole vaja otsida täielikku koondumist, vaid tahetakse teada seisu näiteks pärast kahte või kolme iteratsiooni.

Lisaks algsete keskpunktide valikumeetodite rakendamisele on võimalik keskpunkte lugeda ka failist. Antud võimalust võib kasutada näiteks koos iteratsioonide arvu määramisega – tehakse algselt tehakse ära mingi kindel arv iteratsioone, salvestatakse keskpunktid faili ja järgmine kord jätkatakse klasterdamist salvestatud keskpunktide pealt.

Kõik funktsioonid koos kirjeldustega on esitatud tabelis 5.

Kasutusjuhend

Käesolevas alapeatükis antakse juhendid loodud tarkvara installeerimiseks ja edukaks kasutamiseks.

Süsteeminõuded

Selleks, et loodud tarkvara kasutada on vaja gcc kompilaatorit, mis toetab C11 standardit ja make, mis automatiseerib rakenduse ehitamist.

Rakendust on testitud nii Windows 10 kui ka Scientifix Linux 6.5 platvormidel.

Kompileerimiseks kasutati GNU gcc kompilaatorit. Täpsemalt gcc (GCC) 4.4.7 20120313 (Red Hat 4.4.7-4) ja GNU Make 3.81.

Rakenduse installeerimine

Selleks, et rakendus installeerida, tuleb kõige pealt navigeerida käsureal lahtipakitud koodi kausta. Olles kaustas, kus kus paikneb fail nimega „Makefile“, kirjutada käsureal „make“ ja vajutada ENTER. Ekraanile peaks ilmuma midagi sarnast joonisele 19. Kui seda ei

toimunud, siis ollakse kas vales kaustas, on miskit valesti süsteemi keskkonnamuutujatel või pole süsteeminõuded täidetud.

```
$ make
gcc -Wall -pedantic -ansi -std=c11 -Wextra -O3  -lm -c auxfunctions.c -o auxfunctions.o
gcc -Wall -pedantic -ansi -std=c11 -Wextra -O3  -lm -c main.c -o main.o
gcc -Wall -pedantic -ansi -std=c11 -Wextra -O3  -lm -c initmethods.c -o initmethods.o
gcc -Wall -pedantic -ansi -std=c11 -Wextra -O3  -lm -c lloyd.c -o lloyd.o
gcc -Wall -pedantic -ansi -std=c11 -Wextra -O3  -lm -c elkan.c -o elkan.o
gcc -Wall -pedantic -ansi -std=c11 -Wextra -O3  -lm -c hamerly.c -o hamerly.o
gcc -Wall -pedantic -ansi -std=c11 -Wextra -O3  -lm -c macqueen.c -o macqueen.o
gcc -Wall -pedantic -ansi -std=c11 -Wextra -O3  -lm -c hartiganwong.c -o hartiganwong.o
gcc -Wall -pedantic -ansi -std=c11 -Wextra -O3  -lm -c metrics.c -o metrics.o
gcc -Wall -pedantic -ansi -std=c11 -Wextra -O3  -lm auxfunctions.o main.o initmethods.o lloyd.o elkan.o hame
rly.o macqueen.o hartiganwong.o metrics.o -o clbin
```

Joonis 19. Tarkvara installeerimine.

Windowsi operatsioonisüsteemides peaks eduka kompileerimise korral ilmuma samasse kausta *clbin.exe* ning Linuxil baseeruvatel operatsioonisüsteemidel *clbin.*

Kui nüüd käivitada programmi ilma parameetriteta, kirjutades käsureale „*./clbin“* või Windows operatsioonisüsteemil „*clbin.exe“*, siis peaks käsureale ilmuma tekst nagu on kujutatud joonisel 20.

```
$ ./clbin.exe
Help menu. Argument and its value must be separated by a space:
-h  - help menu
-f  - input data points file name
-o  - output file name
-ci - centers input file name
-a  - algorithm choice (lloyd, elkan, hamerly, macqueen, hartigan, closest)
-i  - initialization method choice (kpp, forgy, partition, furthest)
-m  - metric choice (euclidean, manhattan)
-k  - clusters count, -ci flag has a higher priority
-s  - random seed nr, otherwise uses current time as the seed. Used to confirm clustering results
-n  - iteration count (default 100)
```

Joonis 20. Programmi käivitamine. Kuvatakse abimenüü.

Selleks, et testida, et kas programm töötab võib jooksutada programmi argumentidel:

```
-a elkan -f a3.txt -m euclidean -o tulemus -k 50
```

Kui käsureale ilmus midagi sarnast nagu joonisel 21 ning kausta tekkis juurde kaks faili nimedega „tulemus“ ja „tulemus_centers“ siis see tähendab, et programm töötab nagu vaja.

```
$ ./clbin -a elkan -f a3.txt -m euclidean -o tulemus -k 50
Using Euclidean metric
Data loaded successfully
n: 7500, d: 2
Data points loading time: 0 seconds 15 milliseconds
Using random seed: 1463023322
Using Forgy center initialization
Center initialization time: 0 seconds 0 milliseconds
k: 50
Doing Elkan clustering
Elkan stopped after 42 iterations
Clustering time: 0 seconds 31 milliseconds
Total Within Group Distance (TWGD) for clustering is: 14935158.332890
Within Cluster Sum of Squares (WCSS) for clustering is: 39604064407.044525
Done with clustering, start writing output to a file
Successfully wrote objects assignments to file
Successfully wrote cluster centers to a file
```

Joonis 21. Rakenduse töötamise katsetamine.

Tabel 5. Rakenduse parameetrite kirjeldused.

Parameeter	Kirjeldus

-h	Kuvab abimenüü.
-f [failinimi]	Argumendiks on failinimi, kust loetakse andmeobjektid (kohustuslik).
-o [failinimi]	Argumendiks on failinimi, kuhu kirjutatakse klastrite indeksid, kuhu andmeobjektid kuuluvad. Keskpunktid kirjutatakse faili nimega [failinimi]_centers. (Kui parameetrit –o pole kasutatud, siis kirjutatakse objektide klastritesse kuuluvused standardväljundisse).
-ci [failinimi]	Argumendiks on failnimi, kust loetakse klastrite keskpunktid (valikuline). Kui –ci on kasutusel, siis ignoreeritakse –k ja –i parameetreid.
-a [algoritm]	Klasterdusalgoritm, mida kasutatakse. Valik: lloyd (Lloyd), elkan (Elkan), hamerly (Hamerly), macqueen (MacQueen), hartigan (Hartigan-Wong), closest (tagastab punktidele lähimad klastrid). Vaikeväärtusena kasutatakse Lloydi algoritmi.
-i [algoritm]	Algsete keskpunktide valimise meetod. Valik: kpp (k-means++), forgy (Forgy), partition (jagamise meetod), furthest (kaugem enne meetod), firstn (keskpunktideks valitakse k esimest andmeobjekti). Vaikeväärtusena kasutatakse Forgy algoritmi
-m [meetrika]	Meetrika, mida kasutatakse. Valik: euclidean (eukleidiline kaugus), manhattan (Manhattani kaugus). Vaikeväärtusena kasutatakse eukleidilist kaugust.
-k [täisarv]	Argumendiks on klastrite arv (kohustuslik, v.a. juhul kui on kasutatud parameetrit ci).
-s [täisarv]	Argumendiks on pseudojuhuslike arvude generaatori seeme. Vaikeväärtuseks on süsteemiaeg.
-n [täisarv]	Argumendiks on iteratsioonide arv, pärast mida klasterdusalgoritmide töö lõppeb. Vaikeväärtusena kasutatakse 100.

Sisend- ja väljundfailide formaadid

Antud alapeatükis antakse ülevaade rakenduse poolt töötlevate failide formaatidest. Failide formaadid on hoitud võimalikult loogiliste ja lihtsatena.

Andmeobjektide sisendfail

Andmeobjektide sisenda on kaks täisarvu n (andmeobjektide arv) ja d (objektide dimensionaalsus). Järgmisel n real on igas reas d topelttäpsusega ujukomaarvu:

```
n d
x11 x12 ... x1d
x21 x22 ... x2d
... ... ... ...
xn1 xn2 ... xnd
```

Näide objektide sisendfailist, kus on 3 2-dimensionaalset objekti:

```
3 2
3.9241 -2.9
-3.384 4.799
-0.8012 -0.002
```

Keskpunktide sisendfail

Esimesel real on üks täisarv k (keskpunktide arv). Eeldatakse, et k järgneval real on igas reas d topeltäpsusega ujukomaarvu, kusjuures d on sama mis andmeobjektide faili puhul.

Formaat:

```
k
c11 c12 ... c1d
c21 c22 ... c2d
... ... ... ...
ck1 ck2 ... ckd
```

Näide keskpunktide sisendfailist, kus on 3 2-dimensionaalset keskpunkti:

```
3
2.99 0.148
-2.00 -3.23
-1.729 2.87
```

Tulemuse väljundfail

Kui programmi kasutamisel on määratud parameeter „–o“ (vt tabel 5), siis kirjutatakse klasterdamise tulemus väljundfaili, mis on antud parameetri väärtusena antud.

Tulemuse väljundfailis on n rida, kus igal real on üks täisarv, mis tähistab, mitmendasse klastrisse i-ndas andmeobjekt kuulub. Kui andmeobjekt oli sisendfailis i-ndal real, siis tulemuse väljundfailis on tema kuuluvust tähistav täisarv (i-1)-ndal real, kuna andmeobjektide sisendfaili esimesel real oli kirjeldatud andmeobjektide ja dimensioonide arvud.

Näide lihtsat tulemuste väljundfailist, kus on 3 andmeobjekti klastrite indeksid:

```
0
1
0
```

Keskpunktide väljundfail

Keskpunktide tulemuste väljundfaili kuju on sama, mis keskpunktide sisendfailil. Antud samasuse põhjuseks on, et siis on võimalus kasutada antud väljundfaili ka keskpunktide sisendfailina.

Kasutamise näide

Järgnevalt näitame, kuidas on võimalik rakendust kasutada k-keskmiste klasterdamise läbiviimiseks.

Koos programmiga tuleb kaasa andmestik „A3.txt“, mis paikneb programmikoodi kaustas. A3 andmestikku kirjeldatakse peatükis „*5.1 Andmestikud“*. A3 andmestikus on 7500 andmeobjekti, millel igaühel on 2 tunnust.

Oletame, et kasutaja tahab jagada need andmeobjektid k-keskmiste meetodit kasutades 25 klastrisse. Meetrikana soovitakse kasutada eukleidilist kaugust, klasterdusalgoritmina tahetakse kasutada Hartigan-Wongi algoritmi ning algsete keskpunktide valimise meetodina kaugem enne meetodit. Tulemus tahetakse salvestada faili nimega „25_klastrit“

Kui kasutaja on kasutavate meetrika ja meetodite osas oma valikud teinud, siis vaadatakse parameetrite tabelist (vt tabel 5), kuidas argumente ette anda.

Selleks, et lugeda sisse andmestik „A3.txt“, peab andmestiku andma ette parameetriga „-f“ (vt tabel 5) ning parameetri väärtuseks „A3.txt“. Tulemuse faili nimega „25_klastrit“ kirjutamiseks kasutatakse „-o“ parameetrit. Hartigan-Wongi algoritmi rakendamiseks antakse parameetrile „-a“ väärtus „hartigan“. Meetrika kasutamiseks kasutatakse parameetrit „-m“ ning eukleidilise kauguse valimiseks kasutatakse väärtusena „euclidean“. Selleks, et kasutada algsete keskpunktide valimiseks kaugem enne meetodit, antakse parameetrile „-i“ väärtusena ette „furthest“ ja lõpuks, et jaotataks andmeobjektid 25 klastrisse antakse parameetrile „-k“ väärtus „25“.

Lõplik parameetrite jada peaks siis lõpuks välja nägema nagu järgnev koodinäide.

```
-f A3.txt -o 25_klastrit -a hartigan -m euclidean -i furthest -k 25
```

Seega jooksutades programmi järgnevalt peaks ilmuma konsooli väljund, mis sarnaneb joonisel 22 kujutatud programmiväljundiga.

```
./clbin f A3.txt -o 25_klastrit -a hartigan -m euclidean -i furthest -k 25
```

ilmuma midagi sarnast nagu joonisel 22.

Samasse kausta peaksid ilmuma failid „25_klastrit“ ja „25_klastrit_centers“. Failis „25_klastrit“ peaks olema tulemuse väljundfaili formaadis antud klasterduse tulemus ja failis „25_klastrit_centers“ lõplikud keskpunktide asukohad.

```
$ ./clbin -f A3.txt -o 25_klastrit -a hartigan -m euclidean -i furthest -k 25
Using Euclidean metric
Data loaded successfully
n: 7500, d: 2
Data points loading time: 0 seconds 0 milliseconds
Using random seed: 1463028017
Using furthest first for centers initialization
Center initialization time: 0 seconds 0 milliseconds
k: 25
Doing Hartigan-Wong clustering
Hartigan-Wong stopped after 5 iterations
Clustering time: 0 seconds 31 milliseconds
Total Within Group Distance (TWGD) for clustering is: 28682106.224403
Within Cluster Sum of Squares (WCSS) for clustering is: 128603110104.409256
Done with clustering, start writing output to a file
Successfully wrote objects assignments to file
Successfully wrote cluster centers to a file
```

Joonis 22. Programmi kasutamise näidis A3 andmestikul.

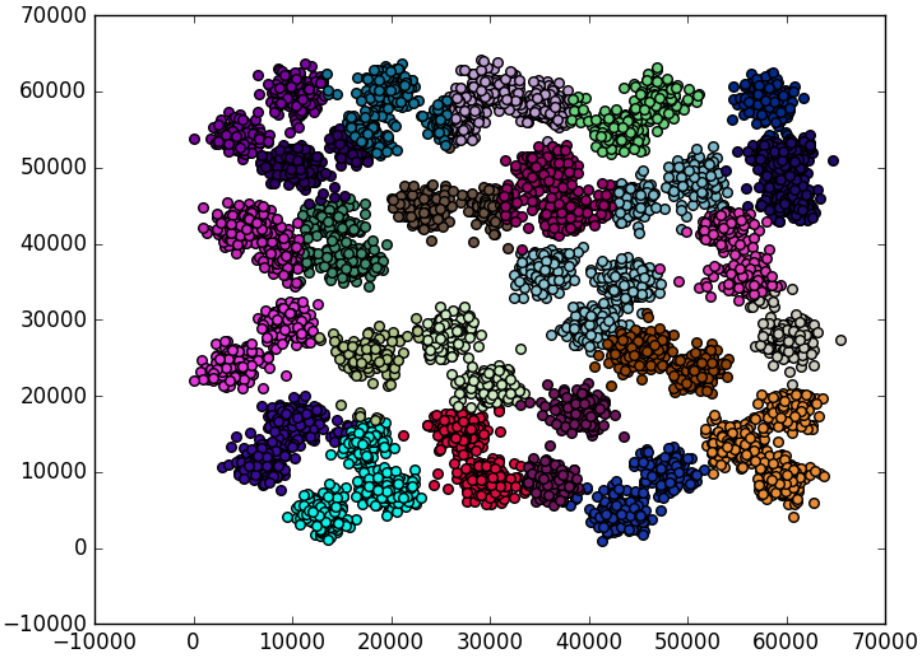
Kasutades lisana antud visualiseerimiseks mõeldud skripti, peaks tulemus sarnanema joonisele 23.

4.4 Rakenduse piirangud

Tarkvara kasutamise korral peab meeles pidama mõningaid järgmiseid piiranguid:

1) Andmeobjektide arvu piirang – koodis on kasutatud andmeobjektide indekseerimise jaoks muutujat size_t, mis 32-bitistel süsteemidel hoiab väärtuseid 0 kuni 2^{32}-1 ja bitistel süsteemidel 0 kuni 2^{64}-1. Seega on maksimaalne võimalik andmeobjektide hulk 32-bitisel süsteemil 2^{32}-1 ja 64-bitistel süsteemidel 2^{64}-1.

2) Andmestiku dimensioonide arvu puhul kehtib sama, mis andmeobjektide puhul.
3) Klastrite arvu puhul kehtib sama, mis andmeobjektide puhul.
4) Andmeobjektide mälushoidmiseks kasutatakse topelttäpsusega ujukomaarvude massiivi[26].

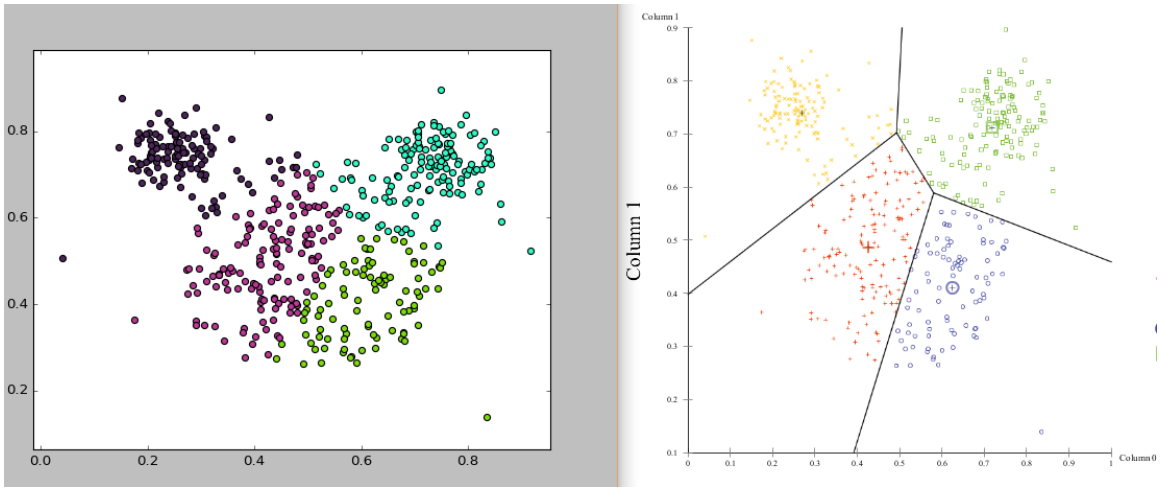


Joonis 23. Näidisklasterduse teel saadud tulemus A3 andmestikul, k = 25.

4.5 Rakenduse kvaliteedi tagamine

Esiteks võrreldi autori poolt loodud rakenduse klasterdamisalgoritmide väljundit ELKI programmi väljundiga. Leiti, et mõlemas programmis saadakse samade algoritmide puhulsarnaseid väljundeid (vt joonis 24). Implementatsioonide jõudlust, mälukasutust ja lähendi leidmise täpsust katsetatakse ka järgnevas peatükis. Samuti kirjutati autori poolt skript, mis annab programmile ette kõikvõimalikke kombinatsioone implementeeritud meetrikatest, algkeskpunktide valikumeetoditest ja algoritmidest ning klasterdab neid kasutades juhuslikult genereeritud andmeid. Testijal on võimalik uurida väljundfaili tulemuste kohta, et näha, kas kõikide sisendparameetrite korral klasterdamine õnnestus

[26] https://en.wikipedia.org/wiki/IEEE_floating_point (viimati vaadatud 12.05.2016)



Joonis 24. Bakalureusetöö käigus leitud rakenduse klasterduse (vasakul) ja ELKI (paremal) klasterduste võrdlus ELKI näidisandmestikul.

5. Implementatsioonide katsetamine

Algoritmide täitmisaja, mälukasutuse ja leitud lähendi kvaliteedi hindamiseks jooksutati antud algoritme mitmesugust andmestike peal. Lähendi kvaliteedi hindamise kriteeriumiks võeti nii ruutvigade keskmised kui ka minimaalsed summad.

Peatüki alguses antakse ülevaade kasutatud andmestikest. Andmestike kirjeldustele järgnevad täitmisaegade, lähendi kvaliteedi ja mälukasutuse mõõtmiseks kasutatavad metoodikad ja vastavad tulemused.

5.1 Andmestikud

Rakenduse ja implementeeritud algoritmide testimiseks on vaja andmestikke, mille peal rakendust jooksutada. Katsetusteks valiti välja neli eri liiki andmestikku.

Andmete üldine kuju

Selleks, et kasutada nimetatud andmestikke peab nad viima kõigepealt rakendusele vastuvõetavale kujule. See tähendab, et objektide sisendfailis peab esimesel real olema kaks täisarvu, failis olevate andmeobjektide arv n ja andmete dimensionaalsus d. Järgneval n real peab olema tühikute või tabulaatoritega eraldatud d reaalarvu.

KDDCUP04Bio andmestik

Esimeseks valitud andmestikuks valiti andmestik, millel on palju andmeobjekte ja keskmine dimensionaalsus. Valitud andmestik on pärit KDD CUP 2004 andmekaeve võistluselt[27]. Andmestikus on 145752 andmeobjekti ning on 74-dimensionaalne. Tunnused näitavad ühe valgujada teistega joondamise sobilikkust.

E-TABM-185 andmestik

Teise andmestiku puhul otsustati valida kõrgedimensionaalse andmestik. Andmestikus valiti E-TABM-185 andmestik[28], kus on 5896 andmeobjekti, mille igal andmeobjektil on 22284 tunnust. Antud andmestik pärineb bioinformaatika valdkonnast.

A3 andmestik

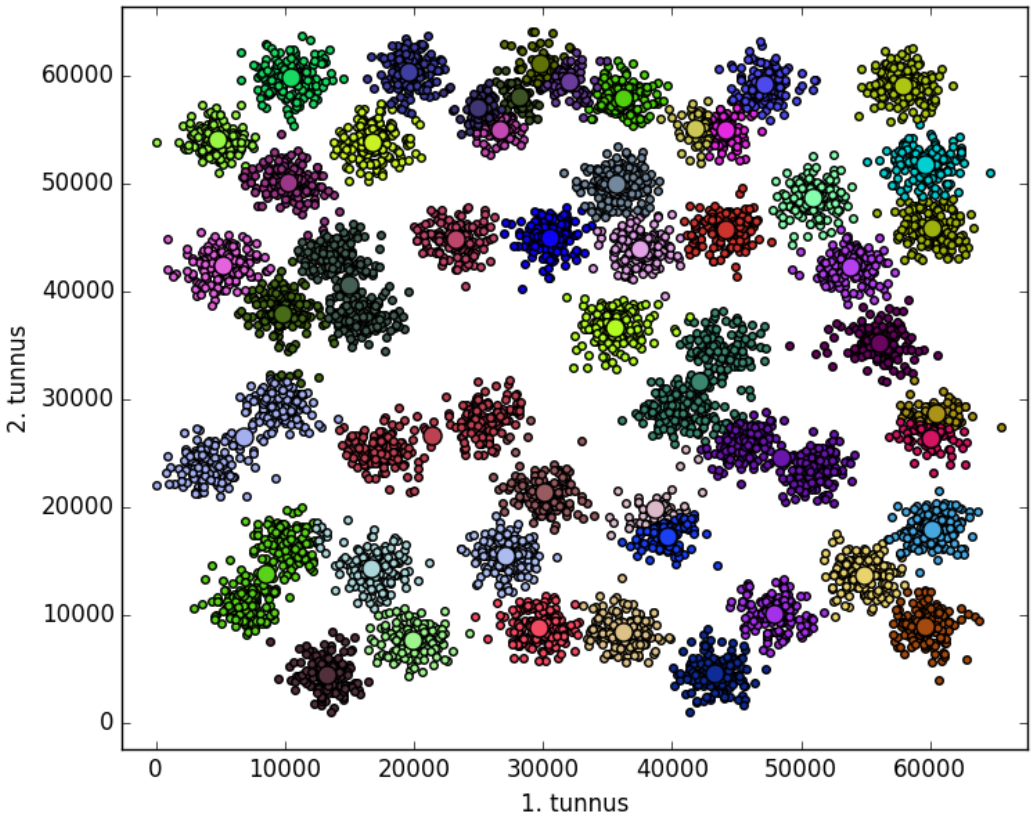
Kolmas andmestik on sünteetiline andmestik [18], mille sisemine struktuur kujutab endast 50 k-keskmiste meetodile sobilikku klastrit (vt joonis 25). Andmestikus on 7500 kahemõõtmelist andmeobjekti. Antud andmestikku kasutatakse ära lähendi kvaliteedi hindamises.

Sünteetilised ühtlase jaotusega andmestikud

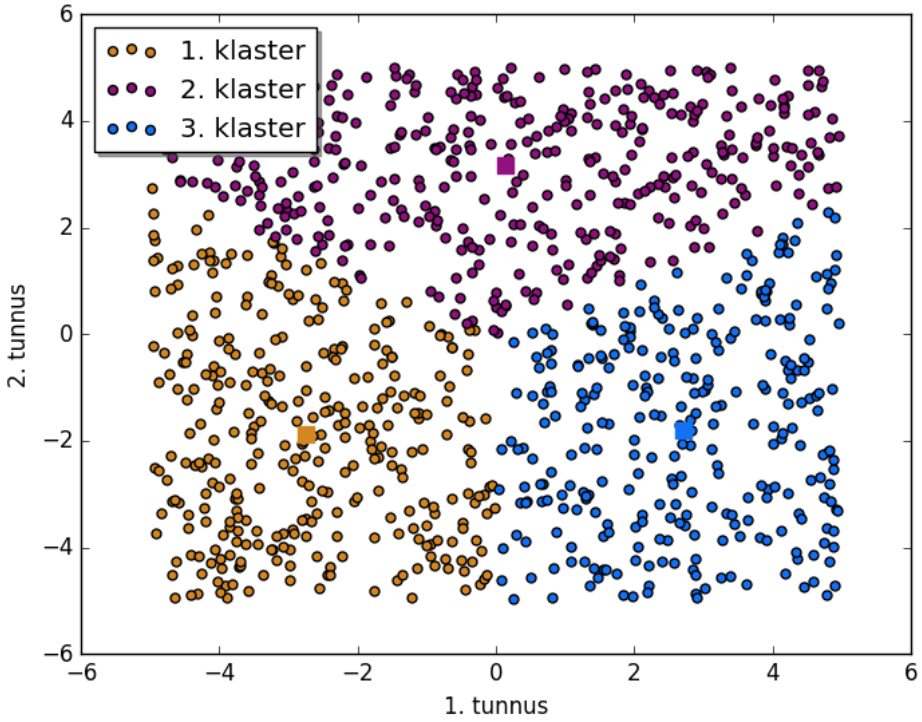
Selleks, et mõõta algoritmide mälukasutuse ja täitmisaja kasvu andmete dimensionaalsuse kasvamise korral, kasutatakse genereeritud ühtlase jaotusega andmestikke (vt joonis 26), mille tunnuste väärtused olid lõigus [0, 1]. Mälukasutuse mõõtmiseks kasutati andmestikke 250 000 ja 500 000 objektiga ning dimensioonidega 32-512. Andmestike genereerimiseks kasutatud skript on lisana kaasa pandud.

[27] http://osmot.cs.cornell.edu/kddcup/

[28] https://www.ebi.ac.uk/arrayexpress/experiments/E-TABM-185/



Joonis 25. A3 andmestiku klasterdus, k = 50.



Joonis 26. k-keskmiste meetodi klasterdus kolme klastrisse ühtlase jaotusega andmestikul n = 1000, d = 2.

5.2 Algoritmide täitmisaeg

Üheks algoritmide efektiivsuse tunnuseks on nende täitmisaeg. Antud alapeatükis kirjeldame kõigepealt metoodikat, millega võrreldakse täitmisaega antud töös implementeeritud algoritmide puhul. Metoodika kirjeldusele järgneb täitmisaja mõõtmiste tulemuste kokkuvõte.

Metoodika

Algoritmide täitmisaja võrdlemiseks jooksutati iga algoritmi vähemalt 5 korda samade argumentidega, muutes seejuures pseudojuhusliku arvugeneraatori seemet. Mõõdetud täitmisajaks võeti kõikide kordade keskmine. Sama andmestiku puhul kasutati kõikide algoritmide jaoks samu seemneid, selleks, et algsed keskpunktid oleksid samad. Täitmisajaks loetakse aega, mis kulus klasterdamisele. Aja kulu, mis läheb andmeobjektide failidest lugemisele ja kirjutamisele, ei arvestata. Kõikide täitmisaegade mõõtmistel kasutati eukleidilist kaugusmõõtu. Tuleb silmas pidada, et ei võrrelda otseselt algoritme, vaid algoritmide implementatsioone. See tähendab, et implementatsioonist tingitud ebaefektiivused võivad algoritmide täitmisaega suurendada. Täitmisaegade arvutamiseks kasutati ära süsteemiaega.

Algkeskpunktide valimise meetoditest keskenduti täitmisaegade võrdlemisel k-means++ ja Forgy meetoditele.

Katsetuste läbiviimiseks kasutati Tartu ülikooli HPC keskuse arvutusklastrit Rocket, mis

kasutab Intel(R) Xeon(R) CPU E5-2660 v2 @ 2.20GHz protsessoreid ja muutmälu maksimaalselt 64GB[29]. Lahendused kasutasid kõik topelttäpsusega ujukomaarve.

Rakenduse kompilaatorina kasutati: `gcc (GCC) 4.4.7 20120313 (Red Hat 4.4.7-4)` ning lähtekood kompileeriti lippudega: `„-Wall -pedantic -ansi -std=c11 -Wextra -O3“.`

Tulemused

Kõigepealt võrreldi algoritmide aegu KDDCUP04Bio andmestikul, kasutades algsete keskpunktide valimiseks k-means++ meetodit. Tulemused on tabelis 6 ja kujutatud joonisel 27. Antud andmestikul ilmneb, et klastrite arvu kahekordistamisel kasvab ligikaudu kõikide algoritmide täitmisaeg samuti kaks korda.

Kõige kiiremaks algoritmiks osutus KDDCUP04Bio andmestiku peal Elkani algoritm, mis võttis 512 klastri juures aega ligikaudu 96 sekundit ning 256 klastri korral 55 sekundit. Elkani algoritm osutus teistest algoritmidest täitmisaja poolest efektiivsemaks alates kaheksast klastrist. Kõige kauem võtsid aega MacQueeni ja Lloydi algoritmide implementatsioonid. Jooniselt 23 on näha, et 512 klastri juures on Elkani ja MacQueeni/Lloydi algoritmide täitmisaegade vahe ligi 20-kordne. Täitmisaja efektiivsuselt teine parim algoritm oli Hartigan-Wongi algoritm, mis 512 klastri juures võttis ligikaudu 9 korda vähem aega võrreldes Lloydi algoritmiga. Hamerly algoritm võttis samas 3,5 korda vähem aega võrreldes MacQueeni ja Lloydi algoritmiga.

k-means++ rakendamisel tuli välja (vt tabel 6), et klasterdamise täitmisaeg vähenes tunduvalt. Näiteks Lloydi algoritmi täitmisaeg k-means++ valikumeetodit kasutades 256 klastri juures vähenes ligikaudu 2 korda.

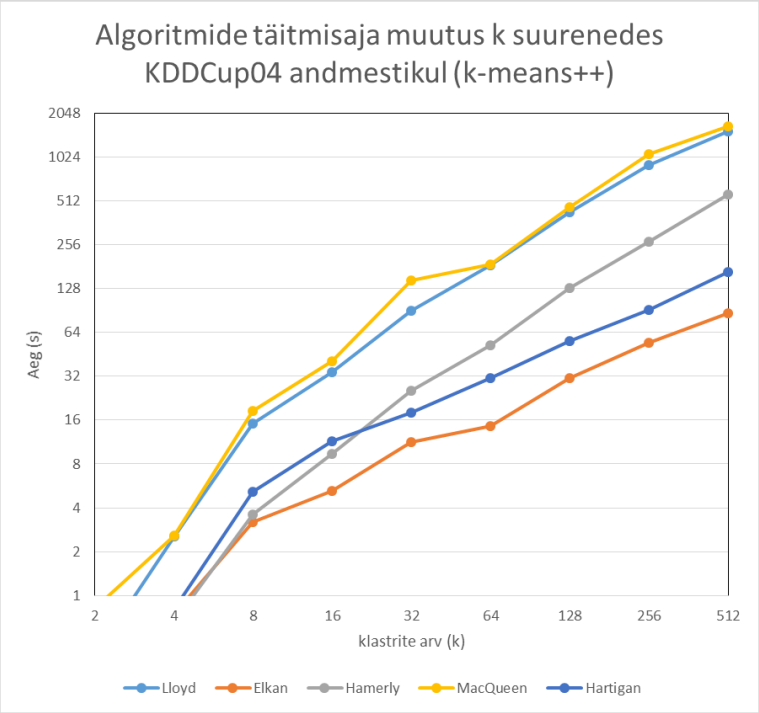
Ilmneb, et antud andmestiku juures on k-means++ algsete keskpunktide valikumeetodile kuluv aeg Elkani, Hamerly ja Hartigan-Wongi algoritmide korral on tunduvalt suurem kui

[29] http://hpc.ut.ee/rocket_cluster

klasterdamisele endale kuluv aeg, mis muudab täitmisaja mõttes k-means++ nende algoritmide jaoks kulukaks. Näiteks Elkani algoritmi korral kulus k-means++ meetodi rakendamisele 256 klastri juures 538 sekundit, aga klasterdamine ise võttis ainult 55 sekundit ning ilma k-means++ meetodita võttis klasterdamine 86 sekundit. Seega võttis k-means++ meetodi enda rakendamine ligikaudu 10 korda rohkem aega. Praeguse andmestiku juures tuli täitmisajaliselt kasuks k-means++ rakendamine ainult Lloydi andmestikul.

Kõrgedimensionaalse E-TABM-185 andmestiku täitmisaegade tulemused on kuvatud joonisel 28 ja tabelis 9. Antud andmestiku puhul kulus samuti kõige vähem aega Elkani algoritmi implementatsioonil, võttes 512 klastri juures aega ligikaudu 192 sekundit. Lloydi ja MacQueeni algoritmi implementatsioonide täitmisaeg oli seejuures 4100 sekundit, mis tähendab, et Elkani algoritmi täitmisaeg oli ligikaudu 21 korda väiksem. Hartigan-Wongi algoritmi implementatsioon võttis ligikaudu 6.5 korda vähem aega võrreldes Lloydi implementatsiooniga ja Hamerly algoritmi implementatsioon ligikaudu 2.1 korda vähem aega.

E-TABM-185 andmestikul kasvab enamus algoritmide täitmisaeg klastrite arvu kahekordistamisel suhteliselt vähe kuni 128 klastrini. Elkani algoritmi tööaeg kasvab klastrite arvu kahekordistamisel vahemikus 16-128 ligikaudu 1,2 korda. Pärast 128 klastrit on täitmisaja kasv ligikaudu lineaarses seoses võrreldes klastrite arvuga. Hamerly algoritm on antud andmestikul ainus algoritm, mille täitmisaeg on pea alati võrdeline klastrite arvuga.



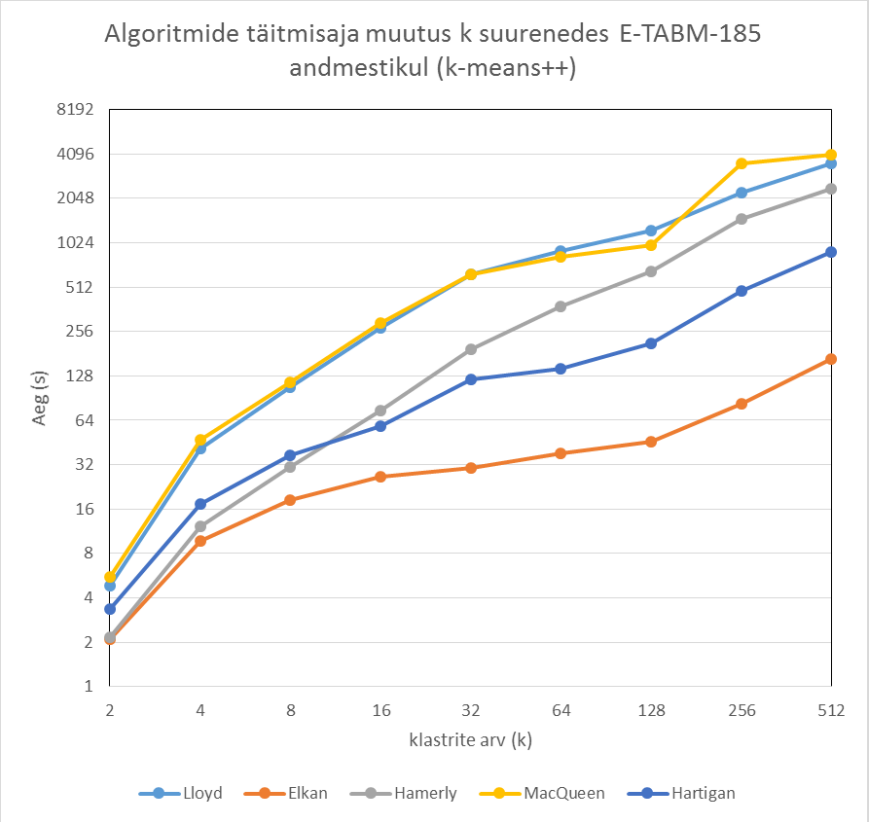
Joonis 27. Algoritmide täitmisaeg KDDCUP04Bio andmestikul, kasutades k-means++ meetodit. Algkeskpunktide valimise aeg pole sisse arvestatud.

Tabel 6. Algoritmide keskmiste täitmisaegade võrdlus KDDCUP04Bio andmestikul klastrite arvu 64, 128 ja 256 puhul. (*) tähendab, et kasutatakse k-means++ meetodit,

muudel juhtudel on kasutatud Forgy meetodit. k-means++ meetodile kuluv täitmisaeg on antud liitmistehte teise argumendina.

Algoritm	k=64	k=64*	k=128	k=128*	k=256	k=256*
Lloyd	**439**	**185 + 168**	**1078**	**429 + 351**	**1813**	**898 + 538**
Elkan	**28**	**15 + 168**	**55**	**31 + 351**	**86**	**55 + 538**
Hamerly	**127**	**53 + 168**	**285**	**129 + 351**	**661**	**269 + 538**
MacQueen	**406**	**187 + 168**	**1250**	**462 + 351**	**1485**	**1068 + 538**
Hartigan	**49**	**31 + 168**	**90**	**56 + 351**	**169**	**92 + 538**

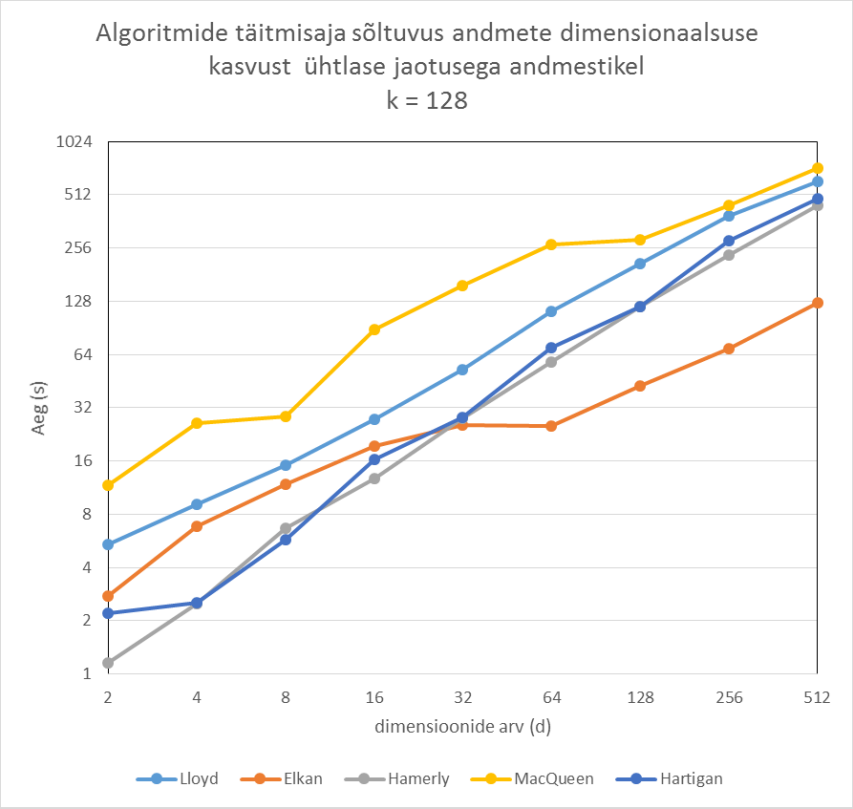
Tabelis 7 on näha, et antud andmestikul on algsete keskpunktide valimine kasutades k-means++ meetodit tunduvalt praktilisem, kui seda oli KDDCUP04Bio andmestikul. Ajalist võitu ei saavutatud 256 klastri juures ainult Hartigani ja Elkani algoritmide puhul.



Joonis 28. Algoritmide täitmisaja sõltuvus klastrite arvust E-TABM-185 andmestikul. k-means++ meetodi aega pole arvestatud.

Tabel 7. Algoritmide keskmiste täitmisajad E-TABM-185 andmestikul klastrite arvu 64, 128 ja 256 puhul. (*) tähendab, et kasutatakse k-means++ meetodit, muudel juhtudel on kasutatud Forgy meetodit. k-means++ meetodile kuluv täitmisaeg on antud liitmistehte teise argumendina.

Algoritm	k=64	k=64*	k=128	k=128*	k=256	k=256*
Lloyd	1505	903 + 34	2559	1228 + 75	3825	2242 + 160
Elkan	59	38 + 34	78	47 + 75	123	73 + 160
Hamerly	722	378 + 34	1018	650 + 75	2101	1490 + 160
MacQueen	1346	824 + 34	2446	982 + 75	5157	3542 + 160
Hartigan	184	143 + 34	308	212 + 75	614	482 + 160



Joonis 29. Algoritmide täitmisaja muutus dimensionaalsuse kasvamisel ühtlase jaotusega andmestikel 128 klastri korral.

Joonisel 29 on näha, et ühtlase jaotusega andmestikel on Hamerly ja Hartigan-Wongi algoritmid on kuni 32 dimensioonini Elkani algoritmist väiksema täitmisajaga. Rohkem kui 32 dimensiooni korral on Elkani algoritm kiirem. Kõige halvemini tuleb toime MacQueeni algoritm.

5.3 Algoritmide lähendi kvaliteet

K-keskmiste optimeerimisülesande sihiks on minimeerida ruutvigade summat ning seetõttu on üheks algoritmi implementatsiooni kvaliteeti määravaks tunnuseks võetud klastrisiseste ruutvigade summa.

Metoodika

Bakalaureusetöö teostaja poolt implementeeritud tarkvara arvutab ja kuvab pärast klasterdamist antud klasterduse ruutvigade summa. Tänu sellele on võimalik siin antud väljundid ära kasutada. Mõõdetakse nii keskmist ruutvigade summat kui ka leitud minimaalset ruutvigade summat. Kuna Lloydi, Elkani ja Hamerly algoritmid leiavad sama lähendi, siis korduvuse vältimiseks on need ühena kokku võetud.

Minimaalse ja keskmise ruutvigade summade hindamiseks kasutasime lisaks E-TABM-185 ja KDDCUP04Bio andmestikule ka andmestikku A3. A3 puhul on tegemist võrdlemisi väikese andmestikuga, kus saab algoritme jooksutada palju kordi vähese aja jooksul.

Tulemused

Tabelites 8-11 on kuvatud k-means++ meetodi rakendamise tulemused ning kui neid võrrelda Forgy meetodiga, siis saab üheselt teha järelduse: nii andmestikul A3 kui ka andmestikul KDDCUP04Bio vähendas k-means++ kasutamine nii minimaalset kui ka keskmist klastrisiseste ruutvigade summat tunduvalt. Minimaalsete ruutvigade vähenemine tähendab, et k-means++ kasutades leiti paremad lähendid kui Forgy meetodiga.

E-TABM-185 (vt tabel 12) andmestikul vähendas k-means++ meetodi rakendamine keskmist ruutvigade summat tunduvalt vähem kui ülejäänud kahel andmestikul. E-TABM-185 andmestikul leidis k-means++ 64 ja 128 klastri juures MacQueeni ja Lloydi algoritmi kasutades kehvemad minimaalsed ruutvigade summad kui Forgy meetodiga. 256 klastri juures leidis k-means++ parema tulemuse.

Kõikide andmestike katsetuste juures leidis Hartigan-Wongi algoritm paremad keskmised ning minimaalsed lähendid võrreldes Lloydi ja MacQueeni algoritmidega.Tabel 8. Klastrisiseste ruutvigade summade sõltuvus algoritmist ja algsete keskpunktide valimise meetodist A3 andmestikul. (*) tähistab k-means++ meetodi kasutamist, teistel juhtudel kasutatakse Forgy meetodit. Tabeli väärtused on klastrisiseste ruutvigade summa.

Algoritm	Keskmine	Minimaalne	Keskmine*	Minimaalne*
Lloyd	**49 542 470 609**	**36 528 869 393**	**40 877 614 620**	**31 973 821 438**
MacQueen	**49 256 159 152**	**36 525 535 653**	**40 841 353 079**	**31 973 740 255**
Hartigan	**48 929 036 392**	**36 459 205 051**	**40 463 043 395**	**31 970 944 699**

Tabel 9. Keskmised klastrisiseste ruutvigade summad KDDCUP04Bio andmestikul. (*) tähistab k-means++ meetodi kasutamist, teistel juhtudel kasutatakse Forgy meetodit.

Algoritm	k=64	k=64*	k=128	k=128*	k=256	k=256*
Lloyd	**265 149 063 419**	**183 750 361 738**	**239 221 885 884**	**152 345 122 675**	**202 735 575 929**	**130 106 440 072**

MacQueen	**265 196 732 451**	**183 752 391 602**	**238 593 367 752**	**152 328 072 593**	**216 816 609 809**	**130 119 033 955**
Hartigan	**265 127 994 520**	**183 624 989 852**	**238 876 901 789**	**152 263 038 544**	**156 428 787 924**	**129 881 027 383**

Tabel 10. Minimaalsed klastrisiseste ruutvigade summad KDDCUP04Bio andmestikul. (*) tähistab k-means++ meetodi kasutamist, teistel juhtudel kasutatakse Forgy meetodit.

Algoritm	k=64	k=64*	k=128	k=128*	k=256	k=256*
Lloyd	**265 029 901 201**	**181 670 242 831**	**236 657 225 658**	**152 134 222 524**	**146 168 755 063**	**129 849 009 573**
MacQueen	**265 052 053 072**	**181 674 363 884**	**236 657 225 658**	**151 525 115 427**	**216 573 924 465**	**129 959 303 688**
Hartigan	**264 839 447 616**	**181 211 474 598**	**236 099 973 453**	**152 028 054 566**	**141 334 803 164**	**129 798 800 365**

Tabel 11. Keskmised klastrisiseste ruutvigade summad E-TABM-185 andmestikul. (*) tähistab k-means++ meetodi kasutamist, teistel juhtudel kasutatakse Forgy meetodit.

Algoritm	k=64	k=64*	k=128	k=128*	k=256	k=256*
Lloyd	**165 995 966**	**165 936 167**	**151 324 116**	**151 218 156**	**138 651 362**	**138 123 214**
MacQueen	**166 055 240**	**165 857 298**	**151 389 667**	**151 064 257**	**138 685 139**	**137 818 659**
Hartigan	**165 263 744**	**164 715 900**	**149 656 494**	**148 909 029**	**135 371 931**	**134 276 173**

Tabel 12. Minimaalsed klastrisiseste ruutvigade summad E-TABM-185 andmestikul. (*) tähistab k-means++ meetodi kasutamist, teistel juhtudel kasutatakse Forgy meetodit.

Algoritm	k=64	k=64*	k=128	k=128*	k=256	k=256*
Lloyd	**165 232 522**	**165 326 382**	**150 227 492**	**150 392 660**	**138 423 341**	**137 126 150**
MacQueen	**165 500 388**	**165 525 289**	**150 596 766**	**151 203 392**	**138 235 152**	**137 818 659**
Hartigan	**164 704 019**	**164 426 117**	**149 194 405**	**148 717 425**	**135 101 631**	**134 098 219**

5.4 Algoritmide mälukasutus

Viimaseks mõõdetud parameetriks oli algoritmide poolt maksimaalselt kasutatav mälu, mis on vajalik, et mõõta algoritmide skaleeruvust suurtele andmestikele.

Metoodika

Algoritmide poolt kasutatav mälu ei ole sõltuvuses objektide omavahelise paiknemisega andmestikes, seega saame mälukasutuse testimiseks kasutada ühtlase jaotusega andmestikke. Seetõttu on mälukasutuse mõõtmine lihtne.

Rocket arvutusklastri peal kasutatakse tööjärjekordade manageerimiseks keskkonda Slurm[30]. Kasutades pärast töö lõppemist käsku `sacct --format JobID, jobname, MaxVMSize` on võimalik leida maksimaalne virtuaalmälu kasutus tehtud tööl.

Tulemused

Mälukasutuse analüüsi tulemused on välja toodud tabelis 13. Testimise tulemusena on näha, et Elkani algoritmi implementatsioon võtab teistega võrreldes tunduvalt rohkem mälu, mis suureneb nii andmeobjektide dimensionaalsuse kui ka klastrite arvu suurendamisel. Teiste algoritmide poolt kasutatud lisamälu võrreldes andmestiku enda suurusega on väike.

Tabel 13. Algoritmide mälukasutus ühtlase jaotusega andmestikel. Mõõtühikuks on bait.

Andmestik	Algoritm	k = 128	k = 256	k = 512	k = 1024
n = 250000, d = 128	**Lloyd**	**258748K**	**259140K**	**259652K**	**260680K**
	Elkan	**509408K**	**760696K**	**1263664K**	**2270840K**
	Hamerly	**262784K**	**263048K**	**263564K**	**264600K**
	MacQueen	**258752K**	**258880K**	**259136K**	**259648K**
	Hartigan	**264608K**	**265000K**	**265516K**	**266552K**
n = 250 000, d = 512	**Lloyd**	**1009648K**	**1010672K**	**1012724K**	**1016820K**
	Elkan	**1261736K**	**1513144K**	**2016732K**	**3026972K**
	Hamerly	**1013556K**	**1014584K**	**1016628K**	**1020724K**
	MacQueen	**1009136K**	**1009648K**	**1010672K**	**1012720K**
	Hartigan	**1015508K**	**1016536K**	**1018592K**	**1022700K**
n = 500 000, d = 128	**Lloyd**	**510836K**	**511092K**	**511604K**	**512632K**
	Elkan	**1014872K**	**1515516K**	**2517568K**	**4524748K**
	Hamerly	**518644K**	**518908K**	**519424K**	**520460K**
	MacQueen	**510700K**	**510832K**	**511088K**	**511600K**
	Hartigan	**522424K**	**522812K**	**523328K**	**524364K**
n = 500 000, d = 512	**Lloyd**	**2011600K**	**2012628K**	**2014676K**	**2018772K**
	Elkan	**2515640K**	**3011048K**	**4020640K**	**6030880K**

[30] http://slurm.schedmd.com/

	Hamerly	**2019416K**	**2020444K**	**2022496K**	**2026584K**
	MacQueen	**2011088K**	**2011600K**	**2012624K**	**2014672K**
	Hartigan	**2023324K**	**2024348K**	**2026400K**	**2030512K**

5.5 Tulemuste kokkuvõte

Katsetatud andmestike peal oli Elkani algoritmi täitmisaeg enamasti madalam võrreldes teiste algoritmidega. Teiseks tööajaliselt efektiivsemaks algoritmiks osutus Hartigan-Wongi algoritm. Kolmandaks Hamerly algoritm. Täitmisajaliselt kõige kehvemaid tulemusi ehk kõige suurema tööajaga algoritmide implementatsioonideks osutusid Lloydi ja MacQueeni implementatsioonid.

Elkani algoritm oli teistest algoritmidest, E-TABM-185 ja KDDCUP04Bio andmestikel, kiirem. Paremaid lähendeid andis Hartigan-Wongi algoritm, mis ei arvesta optimaalsema lahendi otsimisel kauguseid teistesse keskpunktidesse, vaid arvutab klastrisiseste ruutvigade muutu andmeobjekti teise klastrisse määramisel. Ühtlase jaotusega andmestikul tuli esile, et Elkani algoritm muutub Hamerly ja Hartigan-Wongi algoritmidest efektiivsemaks alates 32 tunnustelistes andmetest.

Algsete keskpunktide valimise meetod k-means++ andis nii KDDCUP04Bio kui ka A3 andmestikul märgatavalt paremaid lähendeid kui Forgy meetod, kuid E-TABM-185 andmestikul erilist võitu k-means++ meetodi kasutamisel polnud. Täitmisaegade poolest võttis Elkani, Hartigan-Wongi ja Hamerly algoritmide puhul k-means++ valikumeetodiga klasterdamine rohkem aega kui Forgy meetodit rakendades. Seega peab antud algoritmide puhul mõtlema, et kas soovitakse pigem paremaid lähendeid või madalamat täitmisaega.

Paremaid lähendeid soovides võiks kasutada Hartigan-Wongi algoritmi koos k-means++ algsete keskpunktide valimise meetodiga. Kui soovitakse madalamat täitmisaega ja mälu pole probleemiks, siis oleks tarvilik kasulik kasutada Elkani algoritmi.

6. Kokkuvõte

Töö käigus implementeeriti k-keskmiste optimeerimismeetodil põhinev tarkvara, mis kasutab üheksat erinevat algoritmi ja kahte meetrikat. Seda tarkvara on võimalik kasutada k-keskmiste klasteranalüüsi läbiviimiseks. Rakenduses implementeeritud klasterdusalgoritmide täitmisaegu, mälu ja lähendi kvaliteeti uurides leiti ka implementeeritutest sobilikumad klasterdusalgoritmid, mida kasutada, kui soovitakse kiiret klasterdust või optimaalsemat tulemust.

Edasiste arenduste suhtes on töö teostajal üks peamine idee. Kuna autor on varem tegelenud paralleelsete koodide loomisega nii hajussüsteemidele kui ka graafikakaartidele, siis pakuks töö teostajale huvi implementeerida töös implementeeritud algoritmidest, aga ei pea ka nendega piirduma, ka paralleelseid versioone. Praeguses töös käsitletud algoritmidest oleks seda lihtne teha näiteks Lloydi, Elkani ja Hamerly algoritmide puhul, kuna kauguste arvutamine ja keskpunktide taasarvutamine toimuvad eraldi faasides.

Töö autor polnud enne tööga alustamist klasteranalüüsiga kokku puutunud. Antud töö sooritamisel omandas ta enda mõistes hea arusaama klasteranalüüsi meetoditest ja kasutatavatest algoritmidest. Samuti sai autor mingi kogemuse algoritmide teadusartiklitest implementeerimise osas, mis alguses osutus autori jaoks võrdlemisi keeruliseks.

7. Kasutatud materjalid

[1] MacQueen, J. B. (1967). Kmeans Some Methods for classification and Analysis of Multivariate Observations. 5th Berkeley Symposium on Mathematical Statistics and Probability 1967, 1(233), lk 281–297. http://doi.org/citeulike-article-id:6083430

[2] Lloyd, S. P. (1982). Least Squares Quantization in PCM. IEEE Transactions on Information Theory, 28(2), lk 129–137. http://doi.org/10.1109/TIT.1982.1056489

[3] Daszykowski, M., & Walczak, B. (2010). Density-Based Clustering Methods. *Comprehensive Chemometrics*, *2*, lk 635–654. http://doi.org/10.1016/B978-044452701-1.00067-3

[4] Bradley, P. S., & Bradley, P. S. (1998). Refining Initial Points for K-Means Clustering. *Microsoft Research*, lk 91–99. http://doi.org/10.1.1.44.5872

[5] Aloise, D., Deshpande, A., Hansen, P., & Popat, P. (2009). NP-hardness of Euclidean sum-of-squares clustering. *Machine Learning*, *75*(2), lk 245–248. http://doi.org/10.1007/s10994-009-5103-0

[6] Coluccia, E., & Louse, G. (2004). Applications of Weighted Voronoi Diagrams and Randomization 1 Introduction. *Journal of Environmental Psychology*, *24*(3), lk 329–340.

[7] H. Chang and D.Y. Yeung, Robust path-based spectral clustering. *Pattern Recognition*, 2008. 41(1): lk 191-203.

[8] http://www.statsoft.com/textbook/Cluster-Analysis (viimati vaadatud 12.05.2016)

[9] Elkan, C. (2003). Using the Triangle Inequality to Accelerate k-Means. *Proceedings of the Twentieth International Conference on Machine Learning (ICML-2003)*, lk 147–153. http://doi.org/10.1016/0026-2714(92)90278-S

[10] Celebi, M. E. (2015). *Partitional clustering algorithms*. *Partitional Clustering Algorithms*: lk 54-56. http://doi.org/10.1007/978-3-319-09259-1

[11] Hamerly, G. (2010). Making k -means even faster. *2010 SIAM International Conference on Data Mining (SDM 2010)*, lk 130–140. http://doi.org/10.1137/1.9781611972801.12

[12] Morissette, L., & Chartier, S. (2013). The k -means clustering technique : General considerations and implementation in Mathematica. *Tutorials in Quantitative Methods for Psycology*, *9*(1), lk 15–24

[13] Hartigan, J. a., & Wong, M. a. (1979). A K-Means Clustering Algorithm. *Journal of the Royal Statistical Society*, *28*(1), lk 100–108. http://doi.org/10.2307/234683

[14] Hamerly, G., & Elkan, C. (2002). Alternatives to the k-means algorithm that find better clusterings. *Proceedings of the Eleventh International Conference on Information and Knowledge Management*, *4*(09), lk 600–607. http://doi.org/10.1145/584887.584890

[15] Hochbaum, D. S., & Shmoys, D. B. (1985). A Best Possible Heuristic for the k-Center Problem. *Mathematics of Operations Research*, *10*(2), lk 180–184. http://doi.org/10.1287/moor.10.2.180

[16] http://stefansavev.com/blog/practical-clustering/ (viimati vaadatud 12.05.2016)

[17] Arthur, D., Arthur, D., Vassilvitskii, S., & Vassilvitskii, S. (2007). k-means++: The advantages of careful seeding. *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, *8*, lk 1027–1035. http://doi.org/10.1145/1283383.1283494

[18] I. Kärkkäinen and P. Fränti, "Dynamic local search algorithm for the clustering problem", *Research Report A-2002-6*

[19] H. Chang and D.Y. Yeung, Robust path-based spectral clustering. *Pattern Recognition*, 2008. 41(1): lk 191-203.

[20] Hamerly, G., & Elkan, C. (2002). Alternatives to the k-means algorithm that find better clusterings. *Proceedings of the Eleventh International Conference on Information and Knowledge Management*, *4*(09), lk 600–607. http://doi.org/10.1145/584887.584890

[21] Hochbaum, D. S., & Shmoys, D. B. (1985). A Best Possible Heuristic for the k-Center Problem. *Mathematics of Operations Research*, *10*(2), lk 180–184. http://doi.org/10.1287/moor.10.2.180

[22] http://stefansavev.com/blog/practical-clustering/ (viimati vaadatud 11.05.2016)

Lisad

I. Terminid

Andmeobjekt d-mõõtmeline vektor, kus vektorite elementideks on reaalarvud $(a_1, a_2, \dots, a_d)$	Data object d-dimensional vector, where vector elements are real numbers $(a_1, a_2, \dots, a_d)$
Klaster Andmeobjektide hulk.	Cluster A set of data objects.
Klasteranalüüs Objektide jaotamine klastritesse niimoodi, et ühte klastrisse sattuvad objektid on omavahel rohkem sarnasemad kui teiste klastrite objektidega.	Cluster analysis Task of dividing objects into cluster so, that objects in one cluster are more similar to each other than to objects in other clusters.
Pseudojuhuslike arvude generaator Algoritm, mis võttes endale algväärtusena seemne, genereerib peaaegu juhuslikke jadasid.	Pseudorandom number generator Algorithm, which takes a seed as input, based from which a almost random sequence of numbers is generated
Pseudojuhuslike arvude generaatori seeme Väärtus, mis antakse ette pseudojuhuslike arvude generaatorile, millest lähtudes hakatakse genereerima arvjada.	Pseudorandom number generator's seed. A value, on what pseudorandom number generator bases its generated sequences on.
Tiheduspõhine klasteranalüüs Klasterdusanalüüsi meetod, kus kõrgema tihedusega piirkonnad valitakse klastriteks..	Density-based cluster analysis Method of cluster analysis, where higher density areas are described as clusters.

II. Litsents

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks

Mina, **Joonas Puura**,
(*autori nimi*)

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) enda loodud teose
Tarkvara loomine erinevate k-keskmiste algoritmide rakendamiseks,
(*lõputöö pealkiri*)

mille juhendaja on Jaak Vilo,
(*juhendaja nimi*)

1.1. reprodutseerimiseks säilitamise ja üldsusele kättesaadavaks tegemise eesmärgil, sealhulgas digitaalarhiivi DSpace-is lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
1.2. üldsusele kättesaadavaks tegemiseks Tartu Ülikooli veebikeskkonna kaudu, sealhulgas digitaalarhiivi DSpace´i kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.
2. olen teadlik, et punktis 1 nimetatud õigused jäävad alles ka autorile.
3. kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest tulenevaid õigusi.

Tartus, **13.05.2016**