TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Tarkvarateaduse instituut

Helen Pikkaro 135204IAPB

PARALLELISEERIMISE MÕJU KAASAEGSELE SUURIMA KLIKI LEIDMISE ALGORITMILE, MIS SISALDAB HEURISTILISI VÄRVIMISE TEHNIKAID

Bakalaureusetöö

Juhendaja: Deniss Kumlander
Doktorikraad

Tallinn 2017

Autorideklaratsioon

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

Autor: Helen Pikkaro

22.05.2017

Annotatsioon

Käesoleva töö eesmärgiks on uurida paralleliseerimise mõju suurima kliki leidmise algoritmile, paralleelselt pannakse tööle kuus algoritmi ja kui üks algoritm leiab lahenduse, katkestatakse teiste töö. Sellisel viisil üritatakse suurima kliki leidmise aega kokku hoida.

See tees algab peateema sissejuhatuseks graafiteooria põhimõistete kirjeldamisega. Peatükis 2 tutvustatakse ja kirjeldatakse heuristilisi värvimisalgoritme, mida hiljem kombineeritakse suurima kliki leidmise algoritmiga VColor-u. Kolmandas peatükis tutvustatakse suurima kliki leidmise algoritmi VColor-u ja kirjeldatakse, kuidas seda paralleliseeritakse. Peale seda testitakse juhuslikult genereeritud ja DIMACS graafidel suurima kliki leidmise algoritme kombineeritult peatükis 2 esitatud heuristiliste värvimisalgoritmidega, algoritmid pannakse tööle paralleelselt ja järjestikku ning esitatakse saadud tulemused. Tulemused näitavad suurima kliki leidmisele kulunud aega, kasutatud värvide arvu, suurimat klikki ja läbitud harude arvu. Tulemuste põhjal kirjutati järeldused ja anti ideid tulevasteks uuringuteks.

Tulemustes selgus, et algoritmi paralleliseerimine ei mõjutanud kuidagi harude analüüsimist, leitud suurima kliki tulemust ja värvide kasutamist graafil, küll aga protsessi aeg pikenes. Protsessi võis pikendada paralleliseerimise meetod ise, arvuti ressursside puudus või see, et paralleelselt otsis liiga palju algoritme korraga lahendust.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 52 leheküljel, 5 peatükki, 18 joonist, 18 tabelit.

Abstract

Investigating effects of applying parallel heuristic coloring techniques on modern maximum clique algorithms

The purpose of this thesis is to investigate effects of parallelizing maximum clique algorithms. Six algorithms are going to work in parallel and when one algorithm finds the maximum clique, others are forced to stop searching. This way finding maximum clique should be faster.

This thesis starts from describing basic concepts of graph theory. In chapter 2 heuristic coloring algorithms are introduced, which later in this thesis will be combined with maximum clique algorithm VColor-u. After that maximum clique algorithm is introduced and algorithm parallelization process described. Then algorithms are being tested on randomly generated graphs and DIMACS graphs, six algorithms are tested in parallel and sequentially. The results of our study show the time used by the algorithms to determine the maximal clique, the number of colors used in the process of vertex coloring, maximum clique that was found and number of branches analyzed by maximum clique algorithm. Results were concluded at the end of chapter 4 and there are also some ideas brought up that might become a good start for future researches.

The results of this study show, that parallelization did not affect the results of analyzed branches, maximum clique and number of used colors on DIMACS graphs, but the time of finding maximum clique was longer than the individually working algorithm's time. These results could depend on the fact that computer, what was used for testing, had too poor characteristics or the parallelization process itself was not suitable for this problem. Also, the number of algorithms running in parallel may have been too big. On randomly generated graphs parallel algorithms did not give good results, results were disproportionate on bigger densities, so these results were not analyzed.

The thesis is in Estonian and contains 52 pages of text, 5 chapters, 18 figures, 18 tables.

Lühendite ja mõistete sõnastik

DIMACS	*The Center for Discrete Mathematics and Theoretical Computer Science*
DSatur	*Degree of Saturation* ehk küllastusaste
Greedy	Ahne
IDO	*Incidence Degree Ordering* ehk esinemissageduse astme järgi järjestamise tehnika.
JP	Jones ja Plassmann
LDO	*Largest Degree Ordering* ehk suurima astme järgi järjestamise tehnika.
LF, Largest-First	Suurim esimesena
NP	*Nondeterministic polynomial time*, keerukusklass NP on otsustusülesannete hulk, mida saab lahendada polünomiaalse ajalise keerukusega mittedeterministliku algoritmiga.
PLF, Parallel Largest-First	Paralleelne suurim esimesena
PSL	*Parallel Smallest-Last* ehk paralleelne väiksem viimasena
V2	Versioon 2, kasutusel algoritmi Dsatur juures.
V3	Versioon 3, kasutusel algoritmi Largest-First juures.
VColor-u	*Vertex Color unweighted* ehk tipu värv kaalumata. D. Kumlanderi poolt avaldatud suurima kliki leidmise algoritm [1].

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Graafiteooria sai alguse 1736. aastal, kui L. Euler seda oma töös tutvustas. Pikka aega kasutati graafe, et lahendada meelelahutuslikke probleeme, nagu näiteks Königsbergi sildade ületamine ja ratsukäigud malelaual [2]. Graafiteooria on küll pikka aega kasutusel olnud, aga seda peetakse ikka nooreks, sest seal leidub palju lahendamata probleeme ja kohti, mida teadlased saavad täiendada uute algoritmidega.

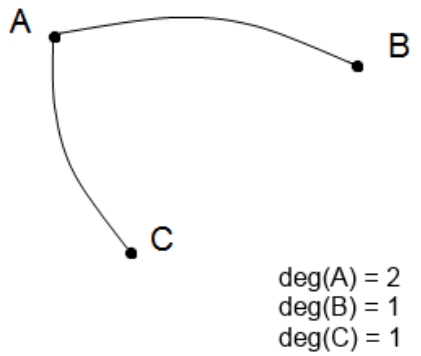
Graaf on objektide vaheliste suhete esitus, mis koosneb kahest rühmast – objektide rühmast, mida kutsutakse tippudeks ja seoste hulgast, mida kutsutakse servadeks. Näitena saab kujutada ühiskonda graafina. Inimesed on objektide rühm ja nende vahelised suhted on seoste hulk. Inimeste vahel leidub palju erinevaid seoseid: pereliige, töökaaslane, sõber, tuttav ja veel palju teisi. Inimesi saab esitada graafi tippudena ja kui ühel inimesel on mingisugune seos teise inimesega selles samas graafis, siis saab nende vahele tõmmata serva. Kasutades graafe, saame lihtsustada reaalseid probleeme matemaatilise mudelina, kus näidatakse ainult probleemi tuuma. Tänapäeval rakendatakse graafiteooriat paljudes erinevates valdkondades, et lahendada keerulisi probleeme ja seetõttu on oluline uurida graafiteoorias leiduvaid probleeme, et muuta seda paremaks.

1.1 Graafiteooria põhimõisted

Graaf on struktuur, mille abil saab koostada objektide hulgas esinevaid seoseid. Graaf (ehk suunamata graaf) G koosneb tippude hulgast V ja servade hulgast E [3]. Lihtgraaf on suunamata graaf, milles ei ole ühtegi silmust (serv, mis ühendab tippu iseendaga) ja kordseid kaari tippude vahel. Selles töös kasutatakse eksperimentides lihtgraafe.

Kaks tippu on seotud külgnevussuhtega, kui ühest tipust läheb kaar teise tippu. Öeldakse ka, et tipud külgnevad [4].

Suunamata graafis on tipu aste $deg(v)$ servade arv, mille otstipuks ta on, kusjuures servi, millel on ainult üks otstipp, loetakse kahekordselt [5]. Tipu astet kujutatakse Joonisel 1: [6]

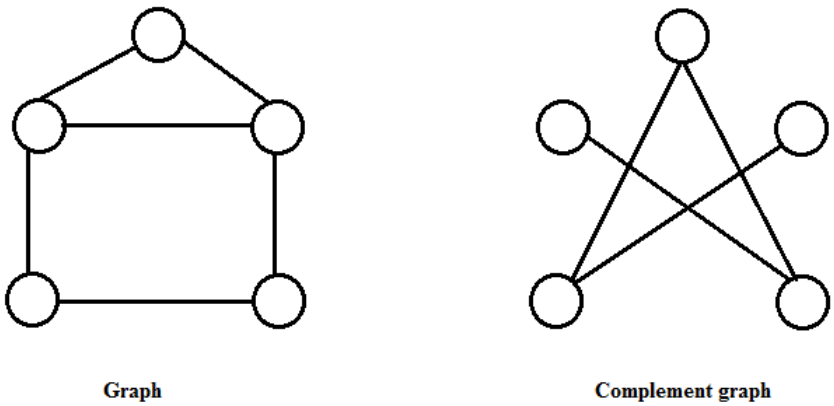


Joonis 1 Tipu aste

Graafi tihedus $g(G)$ on graafi tippude arvu $n = |V|$ ja graafi servade arvu $m = |E|$ suhe. Seda leitakse valemiga [6]:

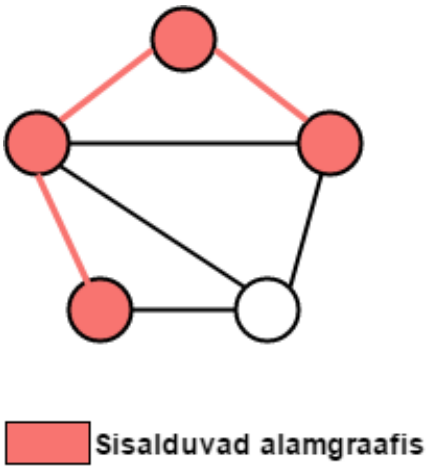
$$g(G) = \frac{2 * m}{n * (n - 1)}$$

Graafi G täiendgraaf on graaf $\overline{G}$, mis omab servi vaid nende tipupaaride vahel, kus graaf G neid ei oma. Graafide G ja $\overline{G}$ ühend on täisgraaf. Täisgraaf on graaf, kus iga kahe erineva tipu vahel on üks serv [3]. Täiendgraaf on kujutatud Joonisel 2: [6]



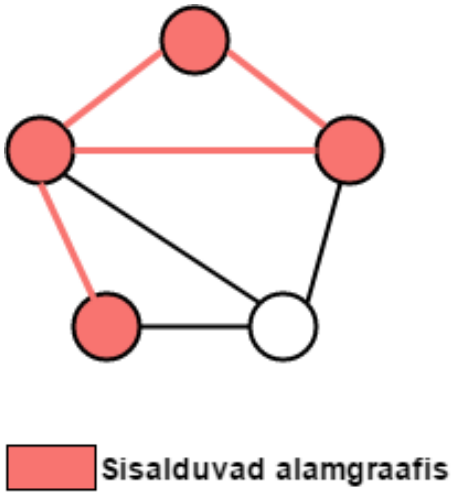
Joonis 2 Täiendgraaf

Alamgraaf $G' = (V', E')$ on graafi G tippude alamhulk vastavate servadega. Kõiki servasid ei pea alamgraafi kaasama, seega võib juhtuda, et graafis G on tipud v_i ja v_j naabrid, aga graafi G alamgraafis nende vahel seost ei ole [6]. Joonis 3 sisaldab näidet alamgraafist.



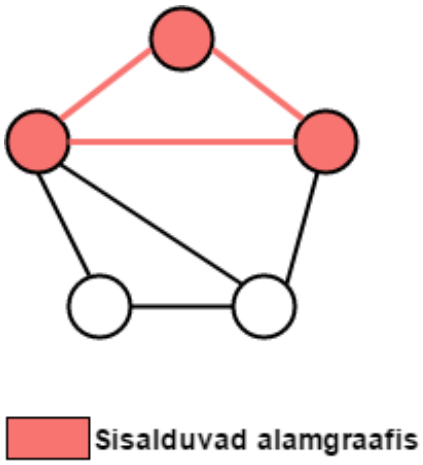
Joonis 3 Alamgraaf

Indutseeritud alamgraaf on graafi G alamgraaf, mis on väljakujunenud graafi G tippude alamhulgast ja kõikidest servadest, mis ühendavad neid alamhulgas leiduvaid tippude paare [7]. Indutseeritud alamgraafi näeb Joonisel 4:



Joonis 4 Indutseeritud alamgraaf

Täielik alamgraaf on graafi G tippude alamgraaf, mille iga tipp on servade kaudu seotud selle alamgraafi teiste tippudega [6]. Täielik alamgraaf on välja toodud Joonisel 5.



Joonis 5 Täielik alamgraaf

Klikk on graafi täielik alamgraaf. Klikki V' graafis G kutsutakse maksimaalseks klikiks, kui seal ei eksisteeri ühtegi teist V'', nii et $V' \subset V''$, ehk klikk ei kuulu ühtegi teise klikki. Graafi klikinumber on suurima kliki suurus graafis G [8].

Suunamata graafi G sõltumatu hulk on selline tippude hulk S, et hulga S iga kahe tipu u ja v puhul serv tippude u ja v vahel puudub [9].

Graafi tippude värvimisel antakse igale tipule värv selliselt, et ühelgi naabertipul poleks sama värv. Graafi kromaatiline arv on minimaalne arv värve, mis on tippude värvimiseks vajalikud. Värviklass on teatud värvidest koosnev tippude alamhulk. Kõik sarnaselt värvitud tipud kuuluvad ühte värviklassi [6].

Algoritmi peetakse heuristilseks, kui see leiab probleemile umbkaudse lahenduse rahuldava aja jooksul. Paljude keeruliste probleemide lahendamiseks kulub väga palju aega selle tõttu, et algoritmid üritavad leida kõige paremat lahendust. Heuristilised algoritmid suudavad leida rahuldava aja jooksul meelepärase vastuse, mitte ei kuluta aega parima vastuse leidmiseks [6].

Algoritmi keerukus on funktsioon $f : N \rightarrow N$, mis seab sisendandmete mahule n vastavusse algoritmi sammude arvu (ajaline keerukus) või kasutatava mälu mahu (mahuline keerukus) [10].

1.2 Ülesandepüstitus

Järgnevalt esitatakse käesoleva bakalaureuse töö jaoks püstitatud eesmärgid:

1. Uurida heuristilisi värvimisalgoritme ja nende eripärasid ning omadusi.
2. Paralleliseerida suurima kliki leidmise algoritm konstrueerides metaalgoritm, mis kasutaks erinevaid heuristilisi värvimisalgoritme.
3. Võrrelda paralleelselt töötavate algoritmide tulemusi järjestikku töötavate algoritmide tulemustega. Tulemuste saamiseks tehakse katseid juhuslikel ja DIMACS graafidel.

1.3 Ülevaade tööst

Käesolev töö sisaldab kolme põhipeatükki. Esimeses peatükis selgitatakse graafi värvimise probleemi ja kirjeldatakse värvimisalgoritme, mida selles töös kasutatakse testimisel. Teises peatükis selgitatakse suurima kliki leidmise probleemi, tutvustatakse suurima kliki leidmise algoritmi VColor-u ja kirjeldatakse, kuidas kuus algoritmi paralleliseeriti. Kolmandas peatükis esitatakse testimisel saadud tulemused, analüüsitakse neid ja luuakse järeldused ning antakse ideid tulevasteks uuringuteks. Testiti suurima kliki algoritme kombineeritult värvimisalgoritmidega juhuslikel ja DIMACS graafidel. Algoritmid pandi tööle paralleelselt ja järjestikku.

2 Värvimisalgoritmid

Graafi värvimise probleemi defineeritakse kui ülesannet määrata graafi tippudele värvid niimoodi, et ükski naabertippude paar ei omaks ühesuguseid värve. Protsessis kasutatud värvide arv peaks jääma nii väikseks kui võimalik [6]. Võib öelda, et kõik sama värvi värvitud tipud kuuluvad sõltumatusse hulka. Graafi värvimise probleem on väga tuntud probleem, selle keerukus on NP-täielik, seetõttu tuleks kasutada selle probleemi lahendamiseks heuristilisi värvimisalgoritme. Heuristiline algoritm ei taga parimat tulemust, kuid see on parimale tulemusele piisavalt lähedal ja selle kiirus on suurem kui täpse algoritmi oma.

Välja on töötatud palju algoritme, mis lahendaksid graafi värvimise probleemi heuristiliselt, aga peamine algoritm graafis värvide määramiseks on Greedy algoritm. Greedy algoritm on üks lihtsamaid heuristilisi algoritme, mida on lihtne teostada ja sellel on väga head tehnilised näitajad [11]. See algoritm pakub võrdlemisi head lahendust vähese aja jooksul. Algoritm käib läbi kõik graafi tipud ja määrab igale tipule väikseima võimaliku värvi, mis ei ole määratud mõnele naabertipule. Ükski naaber ei tohi omada samasugust värvi. Juhul, kui tippu ei ole võimalik värvida mõne olemasoleva värviga, luuakse uus värv ja määratakse see tipule [6]. Joonis 6 kujutab Greedy algoritmi ülesehitust: [6]

```
Let G = (V, E)
For i := 1 to i := Number of vertices:
        c_j = min C, where C = {1, 2, 3, ..., m}
        Try to color v_i with c_j
        If no color was found:
                Create new color class m := m + 1
                Color v_i with c_m
```

Joonis 6 Greedy pseudokood

Tippude värvimise järjekord mängib suurt rolli kogu värvimise protsessis ja mõjutab tugevalt selle kvaliteeti, seetõttu on loodud mitmeid algoritme, mis kasutavad teistsuguseid järjestamise heuristikaid tippude järjekorra määramiseks enne nende värvimist.

Sellised algoritmid põhinevad enamjaolt Greedy algoritmil, aga muudetud on tippude järjestamist, et saavutada paremaid tulemusi [6]. Kõige tuntumad järjestamise heuristikad on:

a. Esmatabamuse (*First-Fit*) järjestamine - kõige algelisem järjestamine. Tipud värvitakse selles järjekorras, kuidas nad ilmuvad graafi esituse sisendis [12].

b. Astme põhjal järjestamine - kasutab tippude järjestamiseks kindlat kriteeriumit ja siis valib välja õige tipu, mida värivda. Selline järjestamine on aeglasem kui esmatabamuse (*First-Fit*) järjestamine, aga värvide kasutamise osas annab palju paremaid tulemusi [6]. Ühed populaarseimad astme põhjal järjestamise heuristikad on:

    - Juhusliku astme järgi järjestamine: graafi tipud värvitakse juhuslikus järjekorras või määratakse igale tipule juhuslikud unikaalsed arvud ja seejärel värvitakse juhusliku astme funktsiooni järgi [6].

    - Suurim aste esimesena (*Largest-First*): graafi tipud värvitakse kahanevalt tippude astmete järgi.

    - Väikseim aste viimasena (*Smallest-Last*): kõigepealt eemaldatakse kõige madalama astmega tipud graafist, siis rekursiivselt värvitakse selle tulemuslik graaf ja lõpuks värvitakse eemaldatud tipud [12].

    - Esinemissageduse astme järgi järjestamine: tipud värvitakse järjest vastavalt sellele, millisel tipul on kõige rohkem värvitud naabreid [6].

    - Küllastusastme järgi järjestamine: värvitakse iteratiivselt värvimata tipud, mille värvitud naabrid kasutavad kõige rohkem erinevaid värve [12].

    - Kombineeritud järjestamine: kombineeritakse erinevaid järjestamise heuristikaid. Näiteks küllastusastme järgi järjestamine kombineeritakse suurim aste esimesena (*Largest-First*) järjestamisega, kus suurim aste esimesena (*Largest-First*) järjestamist kasutatakse siis, kui mõndade tippude küllastusaste on sama [6].

Paljudest populaarsetest algoritmidest on loodud paralleelsed versioonid. Järjestikulised algoritmid teostavad paljusid selliseid ülesandeid järjest, mida saaks teostada samaaegselt ja hoida tänu sellele aega kokku.

Aleksei Kulitškov testis enda töös [6] 56 DIMACS graafil erinevaid värvimisalgoritme ja tõi välja tulemuste tabelid. Parimate tulemuste tabelitest valiti välja 9 algoritmi, mis olid teistest algoritmidest paremad või leidsid graafidel minimaalse võimaliku arvu värve ja seejärel koostati 2 edetabelit. Tabelis 1 järjestati algoritmid edukate kordade arvu järgi ja Tabelis 2 kõikide graafide lahendamiseks kulunud aja järgi.

Tabel 1 Värvimisalgoritmide edetabel, järjestatud edukate kordade arvu järgi.

Algoritm	**Edukate kordade arv**
DSatur-IDO-LDO	54
DSatur V2	52
DSatur-LDO	51
DSatur	49
PSL	46
LDO-IDO	44
PLF	44
LF V3	43
JP V2	29

Tabel 2 Värvimisalgoritmide edetabel, järjestatud aegade summa (ms) järgi

Algoritm	**Aegade summa (ms)**
LDO-IDO	262
LF V3	273
JP V2	576
PLF	1665
DSatur-LDO	1783
DSatur	1811
DSatur V2	1883
DSatur-IDO-LDO	2074
PSL	2778

Nende tabelite põhjal valiti välja 6 värvimisalgoritmi: DSatur V2, Largest-First V3, DSatur-LDO, DSatur-IDO-LDO, LDO-IDO ja Parallel Largest-First.

Neid algoritme kirjeldatakse täpsemalt alljärgnevates alapeatükkides. JP V2 andis küll ajaliselt hea tulemuse, aga edukate kordade arvult oli viimane. PSL ei valitud ajalise viimase koha pärast ja DSatur ei olnud kummaski edetabelis kõrgel kohal, seega otsustati ka see valikust välja jätta.

2.1 DSatur V2

DSaturi algoritmis järjestatakse tipud nende küllastusastmete järgi kahanevalt, see tähendab, et esimesel kohal on tipp, mille naabrid kasutavad suurimat arvu erinevaid värve. Iga iteratsiooni käigus toimub ümber järjestamine. Viigi esinemise korral valitakse tipp suurima arvu värvimata naabrite järgi [1].

DSatur V2 on DSatur'ist tehtud teine versioon. Selles versioonis leitakse kõigepealt suurim klikk graafis ja siis määratakse selle kliki igale tipule erinev värv. Seejärel eemaldatakse need värvitud tipud graafist ja edasi jätkub töö samamoodi nagu DSaturi esialgses versioonis. Täpsemat algoritmi tööd kirjeldab Joonis 7: [6]

```
Let G = (V, E)
U = V
Find the largest clique U* of U
Assign each vertex from U* a possibly low distinct color
U = U \ U*
While U ≠ ∅
        Order vertices by decreasing saturation degree deg(u)
        If a tie, then order by descending number of colored neighbors
        Take the first vertex u from U
        Find the minimum color c_j not used in its neighborhood
        Color u with c_j
        For each neighbor of u:
                Increase its saturation degree if color is not in the
                neighborhood
                Decrease the number of colored neighbors
        Remove the colored vertex from U
```

Joonis 7 DSatur V2 pseudokood

2.2 Largest-First V3

Largest-Frist algoritm järjestab tipud naabrite arvu järgi ja seejärel alustab Greedy värvimisega [6].

Largest-First V2 on täiendatud versioon Largest-First algoritmist. Selles algoritmis iga iteratsiooniga võib värvitud saada rohkem kui üks tipp, näiteks pärast selle tipu värvimist, millel on kõige rohkem naabreid, määrab algoritm sama värvi kõigile teistele tippudele, mille naabertipud ei ole värvitud sama värvi. Lõpuks eemaldatakse need tipud graafist [6].

Kolmas versioon Largest-First algoritmist sarnaneb Largest-First V2-le, aga igas iteratsioonis toimub ümberjärjestamine. Värvitud tipu eemaldamine graafist põhjustab selle eemaldatud tipu naabri servade arvu vähenemist [6]. Joonisel 8 on esitatud Largest-First V3 pseudokood: [6]

```
Let G = (V, E)
U = V
C = { }
While U ≠ ∅
        Order vertices by deg(u) descending
        Add new color c_j to C
        Take the first vertex u from U
        Color u with c_j
        Try to color as many vertices as possible with c_j
        For each neighbor of colored vertices:
                Decrease its degree
        Remove the colored vertices from U
```

Joonis 8 Largest-First V3 pseudokood

2.3 DSatur-LDO

DSatur-LDO on kombineeritud värvimisalgoritm. Algoritm töötab samamoodi nagu DSatur, aga viigi korral lahendatakse konflikt Largest-First algoritmiga. Largest-First algoritm järjestab tipud nende naabrite arvu järgi ja seejärel alustab Greedy värvimisega. Largest-First algoritmi põhiidee on esmajärjekorras hoolitseda nende tippude eest, millel on kõige rohkem naabreid, sest nende seas võib esineda kõige rohkem konflikte.

2.4 DSatur-IDO-LDO

See on kombineeritud algoritm. Selles algoritmis lahendatakse viigid kõigepealt IDO algoritmiga ja seejärel alles jäänud viigid lahendatakse LDO algoritmiga [13]. IDO on täiustatud DSatur algoritm. Tipud järjestatakse nende värvitud naabrite arvu järgi kahanevalt ja kui leidub kaks tippu, millel on sama arv värvitud naabreid, siis kasutatakse juhuslikke arve, et otsustada nende tippude järjekord. Värvimiseks kasutatakse Greedy algoritmi [6].

2.5 LDO-IDO

LDO-IDO on kombineeritud värvimisalgoritm. Selle algoritmi põhiline heuristika on Largest-First algoritm. Viigi korral otsustab IDO heuristika, milline tipp valida. Terviklikult on see algoritm sarnane Largest-First V3 algoritmile, aga koos IDO funktsiooniga, mis tähendab, et esimene järjestus tehakse suurima arvu naabrite järgi ja siis järjestatakse suurima arvu värvitud naabrite järgi [6]. Algoritmi pseudokood on kirjas Joonisel 9: [6]

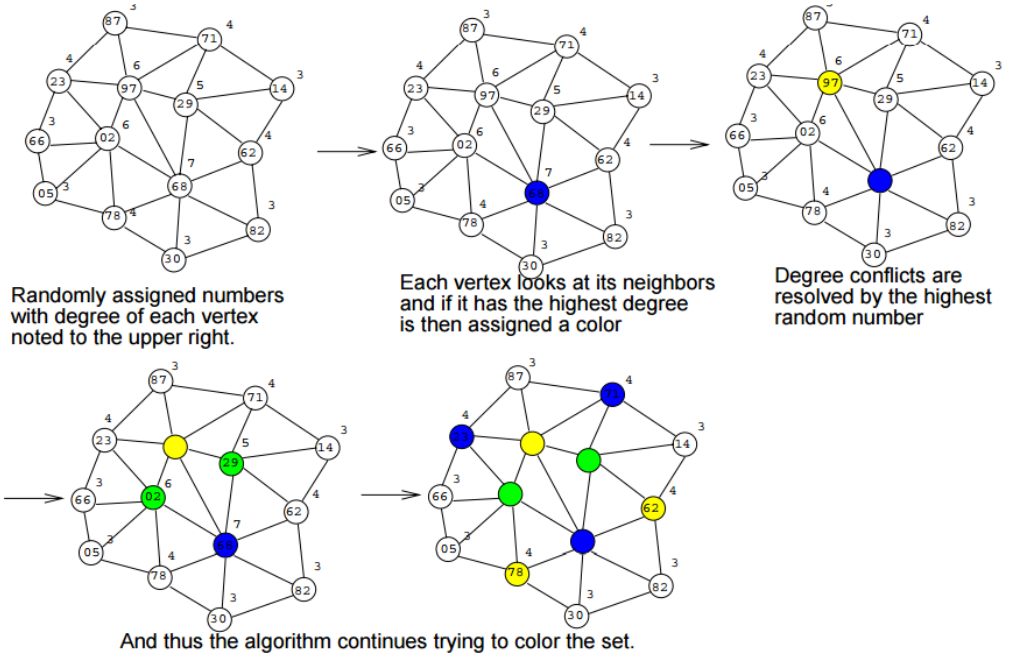
```
Let G = (V, E)
U = V
C = { }
While U ≠ ∅
        Order vertices by deg(u_i) descending
        Then if a tie, order by the number of colored neighbors
        decreasing
        Add new color c_j to C
        Take the first vertex u from U
        Color u with c_j
        Try to color as many vertices as possible with c_j
        For each neighbor of colored vertices:
                Decrease its degree
                Increase the number of colored neighbors
        Remove the colored vertices from U
```

Joonis 9 LDO-IDO pseudokood

2.6 Parallel Largest-First

Parallel Largest-First algoritmi põhjaks on Jones ja Plassmann algoritm, aga heuristiliseks osaks on Largest-First algoritm. JP algoritmi põhiideeks on algoritmi alguses koostada unikaalne kaalude kogum, mida kasutatakse terves algoritmis. Selleks kasutatakse näiteks juhuslikke arve. Samade juhuslike arvude sattumisel kogumisse võetakse appi tippude numbrid. Igas iteratsioonis JP algoritm leiab iseseisva rühma graafis ja seejärel määrab Greedy algoritmi kasutades värvid nendele tippudele. Kogu tegevus toimub paralleelselt [6].

Parallel Largest-First algoritmis leitakse iga tipu suurim aste. Kasutatakse ka juhuslikke arve, mis aitavad lahendada probleemi, kus kahel tipul on sama palju naabreid. Largest-First algoritmis tippude värvimist kirjeldab Joonis 10: [14]



Joonis 10 Largest-First algoritmis tippude värvimise protsess

3 Suurima kliki leidmise algoritmid

Suurima kliki leidmise probleem on probleem, kus otsitakse graafi suurimat võimalikku klikki ehk suurimat täielikku alamgraafi. Seda probleemi peetakse NP-raskeks, sest sellele probleemile on väga raske leida lahendusi tavapäraste meetoditega. Suurima kliki leidmiseks on loodud palju algoritme ja nende täiendatud versioone. Ühed tuntumad algoritmid on aastal 1990 Carraghani ja Pardalose tutvustatud algoritm ja aastal 2002 Östergårdi poolt loodud algoritm. Need algoritmid on tänu oma populaarsusele ja sooritustele saanud põhjaks paljudele täiustatud algoritmidele, mis on loodud selleks, et parandada suurima kliki leidmise kvaliteeti ja vähendada selle aega.

Vanad algoritmid keskendusid rohkem külgnevatele tippudele ja unustasid ära need tipud, mis ei külgne. Kahte mittekülgnevat tippu ei saa lisada ühte klikki, seega kaks tippu erinevatest sõltumatutest hulkadest ei saa olla lahenduses. Kaasaegsed algoritmid sõltuvad suures osas heuristikast, täpsemalt tippude värvimisest. Graafi värvimine laseb ehitada mitmeid sõltumatuid hulkasid ja kasutada kogutud informatsiooni täiendavaid omadusi ning otstarbekalt kasutatud heuristilised lahendused ei suurenda ajakulu dramaatiliselt. Värvimise probleem on NP-täielik, seega ei ole võimalik kasutada täpseid algoritme värviklasside leidmiseks [11]. Üldiselt modernsed algoritmid teostavad eeltööd kogudes ja analüüsides täiendavat informatsiooni enne, kui harude läbitöötamist ehk kliki otsimist.

3.1 VColor-u

VColor-u algoritmi tutvustas Deniss Kumlander oma töös „Some Practical Algorithms to Solve The Maximum Clique Problem“ aastal 2005 [1]. Selle algoritmi lühendi saab dešifreerida „Tipu värvimine kaalumata“ (Vertex Color unweighted). Algoritmi põhiidee oli demonstreerida suurima kliki leidmise probleemi lahendamisel sõltumatute hulkade kasutamise efektiivsust ilma lisatud kiirendavate tehnikateta [6].

Heuristikana kasutatakse Largest-First algoritmi. Seda kasutatakse ainult protsessi alguses, et kindlaks määrata kõik värviklassid ükshaaval, kuni kõik tipud on värvitud. Alles siis, kui kõik värviklassid on vastu võetud, alustab tööd algoritmi peaosa [6]. VColor-u pseudokood on esitatud Joonisel 11: [11]

```
function Main
      Heuristic vertex coloring
      Order vertices that first color classes have the last indexes
      CBC := 0 //The maximum clique's size
      clique (V, 1)
      return CBC
end function
function clique(V, depth)
      If |V| = 0  then
            If depth > CBC then
                  New record – save it
                  CBC := depth
            End if
            Return
      End if
      i := 0
      while i < |V| do
            if depth – 1 + degree (V) ≤ CBC then  // prune
                  return
            i := i + 1  //form a new depth. N(v_i) denotes a
            neighborhood of v_i
            clique (N(v_i)  | ∀v_j : j > i, j ≤ |V|, depth + 1)
      end while
      return
end function
```

Joonis 11 Algoritmi VColor-u pseudokood

Aleksei Kulitškov jättis oma töös [6] sellest algoritmist välja Largest-First heuristika ja asendas teiste värvimisalgoritmidega. Selles töös käitutakse samamoodi, kasutusele võetakse peatükis 2 välja valitud kuus heuristilist värvimisalgoritmi ja lisatakse VColor-u algoritmi.

3.2 Algoritmide paralleliseerimine

Paralleelselt töötavad algoritmid peaksid lahendama probleemi kiiremini, kui seda teeks järjestikku töötavad algoritmid. Paralleelselt töötavas programmis on võimalus katkestada teiste algoritmide töö, kui üks (kõige kiirem) algoritm on jõudnud lahenduseni. Niimoodi ei kulutata aega kõikide lahenduste ootamisele ja kogu protsess peaks olema teoreetiliselt kiirem. Eesmärk on leida võimalikult kiiresti suurim klikk graafis, aga erinevate graafide puhul leiduvad erinevad värvimisalgoritmid, mis aitavad kõige kiiremini vastuse leida, sellepärast on oluline kasutada lahenduse leidmiseks mitut erinevat algoritmi. Joonisel 12 on esitatud programmikood, mida kasutati kuue algoritmi paralleliseerimiseks. Algoritmide paralleliseerimiseks kasutati *Task'e*. Programmis võetakse kuus algoritmi, luuakse kuus *CancellationTokenSource* ja pannakse korraga tööle kuus *Task*'i. *Task*'id käivitavad meetodi *MaxClique()*, kus alustab algoritm oma tööd. Iga algoritmi jaoks on eraldi *Task* ja igale algoritmile määratakse individuaalne *CancellationToken*, mille abil saab anda töötavale *Task*'ile infot, et oleks vaja töö lõpetada. Algoritmid otsivad paralleelselt suurimat klikki graafis ja kui üks algoritm leiab lahenduse, antakse teistele algoritmidele käsk oma töö lõpetada. Seejärel võetakse ette järgmine graaf lahendamiseks ja kõik algoritmid hakkavad uuesti lahendust otsima. Sellisel viisil testiti algoritme juhuslikel ja DIMACS graafidel.

```
private void MaxClique(Type algorithm, Graph graph, int testCaseNumber,
 string testCaseKey, int algorithmNumber, int step,
 KeyValuePair<string, int> testCase, string path, CancellationToken token)
{
	Algorithm alg =
	RunAlgorithm((Algorithm)Activator.CreateInstance(algorithm, graph),
	token);
}
public void Start()
{
	Type algorithm = Algorithms[];
	var cts = new CancellationTokenSource();

	Task.Run(() => MaxClique(algorithm, graph, testCaseNumber,
		testCase.Key, algorithmNumber, step, testCase, path,
		cts.Token), cts.Token),
	};
	var w = Task.WaitAny(ts);
	for (int i = 0; i < cts.Length; i++)
		if (i != w)
			cts[i].Cancel();
}
```

Joonis 12 Algoritmi paralleelselt tööle panemise pseudokood

4 Eksperimendid

Selles peatükis esitatakse algoritmide testide tulemused ja analüüsitakse, kuidas paralleliseerimine mõjutas algoritmide tulemusi. Testimisel kasutati peatükis 2 välja valitud kuute heuristilist värvimisalgoritmi, mis kombineeriti suurima kliki leidmise algoritmiga VColor-u. Neid algoritme testiti juhuslikel ja DIMACS graafidel, algoritmid pandi tööle paralleelselt ja individuaalselt.

Algoritmid teostati C# keeles, testimiseks kasutati programmmi Visual Studio Community 2017 (.Net Framework versioon 4.6). Süsteemi omadused:

- Protsessor: Intel Core i5-6300HQ CPU 2.30GHz (neljatuumaline)
- RAM: 8GB
- Süsteemi tüüp: 64-bit operatsioonisüsteem
- Operatsioonisüsteem: Windows 10 Pro

4.1 DIMACS graafid

Kõigepealt esitatakse DIMACS graafidel saadud tulemused. DIMACS graafidel on igal graafil oma kindel struktuur, mis on vastavuses mõne reaalse probleemiga. Graafid on valitud *Second DIMACS Implementation Challenge* [15] esitatud graafide hulgast, välja valiti sellised graafid, mille lahendamiseks ei kulu rohkem kui tund aega. Tabelis 3 on kirjeldatud kõikide kasutatud 27 graafi andmed: graafi nimi, tippude arv, graafi tihedus ja suurim klikk.

Tabel 3 DIMACS graafide andmed

Graaf	**Tippude arv**	**Tihedus**	**Suurim klikk**
c-fat500-1.clq	500	0,36	14
c-fat500-2.clq	500	0,07	26
c-fat500-5.clq	500	0,19	64
c-fat500-10.clq	500	0,37	126
hamming10-2.clq	1024	0,99	512
hamming6-2.clq	64	0,9	32
hamming6-4.clq	64	0,35	4
hamming8-2.clq	256	0,97	128
hamming8-4.clq	256	0,64	16
johnson16-2-4.clq	120	0,76	8
johnson8-2-4.clq	28	0,56	4
johnson8-4-4.clq	70	0,77	14
keller4.clq	171	0,65	11
MANN_a27.clq	378	0,99	126
MANN_a9.clq	45	0,93	16
p_hat300-1.clq	300	0,24	8
p_hat300-2.clq	300	0,49	25
p_hat300-3.clq	300	0,74	36
p_hat500-1.clq	500	0,25	9
p_hat500-2.clq	500	0,5	36
p_hat700-1.clq	700	0,25	11
p_hat1000-1.clq	1000	0,25	10
san1000.clq	1000	0,5	15
san200_0.7_1.clq	200	0,7	30
san200_0.7_2.clq	200	0,7	18
san200_0.9_1.clq	200	0,9	70
san400_0.5_1.clq	400	0,5	13

Tabelis 4 on esitatud kuue individuaalselt käivitatud algoritmide aegade tulemused ja veerus „Paralleelne algoritm“ kuvatakse selle algoritmi aeg, mille tööd ei katkestatud paralleelselt töötavas programmis. Rohelise värviga on märgitud nende algoritmide ajad, mis leidsid graafil kõige kiiremini lahenduse ja halli värviga on märgitud need paraleelselt tööle pandud algoritmi tulemused, mis jäid graafi lahendamisel kiiruselt teisele või kolmandale kohale. Näiteks graafi c-fat500-1.clq lahendasid kõige kiiremini algoritmid LDO-IDO ja Largest-First V3 (LF V3), ajaga 14 ms ja need on märgitud rohelise värviga, kiiruselt teisele kohale jäi paralleelne algoritm ajaga 24 ms ning see märgiti halli värviga. Kõikide paralleelselt tööle pandud algoritmide aegu kuvab Tabel 11. Paralleelselt tööle pandud algoritmide suurima leitud kliki tulemused on välja toodud Tabelis 12 ja kasutatud värvide arvud on esitatud Tabelis 13, mõlemas tabelis on värviga ära märgitud algoritmid, mis jõudsid esimestena lahendusteni ja tähis (c) on juurde pandud nendele algoritmidele, mille töö katkestati. Järjestikku tööle pandud algoritmide suurima kliki tulemusi näeb Tabelis 15 ja kasutatud värvide arv on esitatud Tabelis 16.

Tabel 4 DIMACS graafidel saadud aegade tulemused.

Graaf	**Aeg (ms)**						
	Paralleelne algoritm	**DSatur V2**	**DSatur-IDO-LDO**	**LDO-IDO**	**PLF**	**DSatur-LDO**	**LF V3**
c-fat500-1.clq	24	50	49	14	103	38	14
c-fat500-2.clq	7	49	74	7	62	40	7
c-fat500-5.clq	21	59	74	13	147	68	12
c-fat500-10.clq	26	125	142	18	201	145	20
hamming10-2.clq	298	5358	5738	180	600054	5029	164
hamming6-2.clq	4	1	3	0	31	1	0
hamming6-4.clq	2	1	1	0	12	1	0
hamming8-2.clq	116	84	85	10	600025	101	11
hamming8-4.clq	107	329	82	3308	575	454	3263
johnson16-2-4.clq	252	540	182	240	206	216	230
johnson8-2-4.clq	20	0	0	0	8	0	0
johnson8-4-4.clq	2	3	3	2	20	3	2

Graaf	Aeg (ms)						
	Paralleelne algoritm	**DSatur V2**	**DSatur-IDO-LDO**	**LDO-IDO**	**PLF**	**DSatur-LDO**	**LF V3**
keller4.clq	450	368	249	729	342	258	710
MANN_a27.clq	126534	600007	443862	600028	75544	436463	599990
MANN_a9.clq	5	6	2	5	85	3	5
p_hat300-1.clq	54	44	48	21	85	46	21
p_hat300-2.clq	1066	1172	916	664	887	857	619
p_hat300-3.clq	1145492	600013	600006	600009	600023	600005	600021
p_hat500-1.clq	494	293	283	183	339	286	179
p_hat500-2.clq	175518	237566	197532	91541	123643	163776	112102
p_hat700-1.clq	1150	993	1016	679	910	955	723
p_hat1000-1.clq	7785	5140	5155	4172	4169	4999	4709
san1000.clq	19321	244489	89893	600023	9371	57604	599997
san200_0.7_1.clq	432	600000	156	600013	600008	9502	3450
san200_0.7_2.clq	116	50	71	47	68	59	89
san200_0.9_1.clq	1193	474143	300915	634	7519	4122	600015
san400_0.5_1.clq	1196	945	592	12808	945	17034	13181

Kõige kiirem algoritm

Paralleelne algoritm jäi kiiruselt II või III kohale

Tabelis 4 näeb, et paralleelselt tööle pandud algoritmid andsid kõige kiiremaid tulemusi kahes graafis, aga nende mõlema graafi puhul leidus teisi individuaalselt tööle pandud algoritme, mis leidsid suurima kliki sama ajaga. Kuuel korral leidis paralleelselt tööle pandud algoritm graafis suurima kliki sellise ajaga, mis jäi kiiruselt teisele kohale. Kuus graafi lahendas paralleelselt tööle pandud algoritm kiiruselt kolmandana. Kõige aeglasemalt leidis paralleelselt tööle pandud algoritm lahenduse samuti kuuel graafil.

Tabelisse 5 kuvatakse algoritmide poolt analüüsitud harude arve. Sarnaselt Tabelile 4 kuvatakse kuue algoritmi individuaalsed tulemused ja veerus „Paralleelne algoritm“ kuvatakse selle algoritmi analüüsitud harude arv, mille tööd ei katkestatud paralleelselt töötavas programmis.

Näiteks algoritm DSatur V2 analüüsis 14 haru suurima kliki leidmiseks graafil c-fat500-1.clq, samale tulemusele jõudis ka paralleelselt töötav algoritm. Kollasega on märgitud need analüüsitud harude arvud, mis olid kõige väiksemad ja halliga on märgitud need paralleelse algoritmi tulemused, mis jäid suuruselt teisele või kolmandale kohale.

Tabel 5 DIMACS graafidel saadud tulemused – analüüsitud harude arv.

Graaf	**Analüüsitud harude arv**						
	Paralleel-ne algoritm	**DSatur V2**	**DSatur-IDO-LDO**	**LDO-IDO**	**PLF**	**DSatur-LDO**	**LF V3**
c-fat500-1.clq	14	14	14	14	127	14	14
c-fat500-2.clq	26	26	26	26	457	26	26
c-fat500-5.clq	64	64	64	64	18999	64	64
c-fat500-10.clq	126	126	126	126	126	126	126
hamming10-2.clq	512	512	512	512	557581449	512	512
hamming6-2.clq	32	32	32	32	519	32	32
hamming6-4.clq	352	340	352	426	398	307	426
hamming8-2.clq	128	128	128	128	651326718	128	128
hamming8-4.clq	215	165371	215	2774392	230199	252410	2774392
johnson16-2-4.clq	353522	887871	353522	489355	353522	401470	489355
johnson8-2-4.clq	37	38	37	59	50	38	59
johnson8-4-4.clq	364	364	18	1171	1882	554	1171
keller4.clq	435550	435550	277698	961304	506491	278088	963672
MANN_a27.clq	8110992	379963666	47164066	254997755	8142781	47164066	326414646
MANN_a9.clq	5224	5224	2171	4093	314	2171	4156
p_hat300-1.clq	25193	25170	26001	23352	25208	25193	23068
p_hat300-2.clq	394533	761014	630453	411228	669518	553779	394533
p_hat300-3.clq	385804658	680474792	436029727	405330292	512844991	389842482	467821907
p_hat500-1.clq	268503	268503	278488	254312	249660	272261	261465

Graaf	Analüüsitud harude arv						
	Paralleel-ne algoritm	**DSatur V2**	**DSatur-IDO-LDO**	**LDO-IDO**	**PLF**	**DSatur -LDO**	**LF V3**
p_hat500-2.clq	45868870	134475894	112346599	48253356	63008021	87977525	59493075
p_hat700-1.clq	977854	1039907	1104598	893465	907068	986604	977854
p_hat1000-1.clq	6556251	6961846	7107052	6539346	6232627	6626485	6556251
san1000.clq	1329301	36642745	53019726	34987626	1329301	40504278	31059470
san200_0.7_1.clq	87844	2240339620	87844	1543532363	2790035880	10344610	1448106
san200_0.7_2.clq	35546	3502	18488	23459	3235	9596	35546
san200_0.9_1.clq	995610	358776158	285335597	995610	65724465	1160105	43718380
san400_0.5_1.clq	267929	487406	267929	4783553	448481	22985290	8623043

Väikseim analüüsitud harude arv

Paralleelne algoritm jäi oma tulemusega II või III kohale

Nendes tulemustes on näha, et 18 korral suutis paralleelselt tööle pandud algoritm leida väikseima analüüsitud harude arvu, kusjuures nendest kolmel korral oli paralleelne algoritm ainus algoritm, mis analüüsis kõige vähem harusid, ülejäänud kordadel leidus veel mõni teine algoritm, mis läbis sama palju harusid graafis. Ühel graafil jäi paralleelselt töötav algoritm analüüsitud harude arvuga teisele kohale. Kolmanda koha saavutas viiel graafil ja kahel graafil oli paralleelselt töötava algoritmi analüüsitud harude arv kõige suurem.

Tabelisse 6 leiti Tabelite 4 ja 11 põhjal iga algoritmi kohta nende kordade arv, kui algoritm leidis kõige kiiremini suurima kliki. Tabelis 4 loeti kokku iga individuaalse algoritmi edukad korrad, kui algoritm saavutas kõige kiirema aja, näiteks DSatur V2 leidis individuaalsel käivitamisel ühel korral kõige kiiremini lahenduse. Tabelist 11 leiti iga paralleelse algoritmi alamosa kohta need korrad, kus algoritm jõudis graafil lahenduseni esimesena.

Meeles tasub pidada, et paralleelses programmis leidis igal graafil ainult üks algoritm tulemuse, teiste töö katkestati pärast esimese tulemuse leidmist. Individuaalselt töötavate algoritmide puhul võis ühel graafil mitu algoritmi leida tulemuse sama kiirusega.

Tabel 6 Mitmel korral algoritm leidis kõige kiiremini lahenduse, DIMACS graafid

	Kordade arv, kui algoritm leidis graafil kõige kiiremini lahenduse	
Algoritm	**Individuaalne käivitamine**	**Paralleelse algoritmi alamosa**
DSatur V2	1	6
DSatur-IDO-LDO	7	6
LDO-IDO	13	6
Parallel Largest-First	3	4
DSatur-LDO	2	1
Largest-First V3	11	4

Võrreldes individuaalse käivitamisega töötasid paralleelse algoritmi alamosana paremini DSatur V2 ja Parallel Largest-First. Dsatur V2 leidis kuuel korral kõige kiiremini tulemuse paralleelse algoritmi alamosana, individuaalselt töötades ainult ühel korral. Parallel Largest-First paralleelne tulemus erines ühe võrra individuaalsest tulemusest, saades 4 korral kõige kiiremini lahenduse. DSatur-LDO leidis paralleelselt kõigest ühel korral kõige kiiremini lahenduse, iseseisvalt töötades aga kahel korral. Võib öelda, et DSatur-LDO oli selles tabelis välja toodud kordade arvu järgi üks nõrgemaid algoritme nii individuaalselt kui ka paralleelse algoritmi alamosana. Individuaalsel käivitamisel oli kõige edukam LDO-IDO, mille kiireimate kordade arv oli 13, paralleelselt oli see arv 6. Largest-First V3 oli iseseisvalt töötades kordade arvu järgi teisel kohal, leides 11 korral kõige kiiremini lahenduse, paralleelselt sai see algoritm kiireima tulemuse 4 graafil.

4.2 Juhuslikud graafid

Järgnevalt esitatakse juhuslikult genereeritud graafidel saadud tulemused. Need testid annavad ülevaate algoritmi üldisest jõudlusest. Joonisel 13 on esitatud funktsioon, mida kasutati juhuslike graafide genereerimiseks: [6]

```
public static Graph GenerateGraph(int nodes, double density)
{
	if (density < 0 || density > 1)
		throw new Exception("0 <= density <= 1");
	int numberOfEdges = Convert.ToInt32(Math.Round(nodes *
	(nodes - 1) * density / 2, 0));

	var graph = new Graph
	{
		Values = new bool[nodes, nodes],
		Edges = numberOfEdges
	};

	var random = new Random();
	Thread.Sleep(40);
	var random2 = new Random();

	int x, y;
	for (int i = 0; i < numberOfEdges; i++)
	{
		do
		{
			x = random.Next(0, nodes);
			y = random2.Next(0, nodes);
		} while (x == y || graph.Values[x, y]);
		graph.Values[x, y] = true;
		graph.Values[y, x] = true;
	}
	return graph;
}
```

Joonis 13 Juhusliku graafi genereerimise funktsioon

Graafi tihedus määrati vahemikus 10% kuni 90%. Igale tihedusele valiti tippude arvu vahemik, mis selle tiheduse juures annaks mõistliku töö aja. Sellega igas tiheduses teostati 11 testi erinevate tippude arvuga. Tihedustel 60%, 70%, 80% ja 90% tekkisid ebaproportsionaalsed tulemused, mida oleks vaja eraldi uurida, seega otsustati need sellest tööst välja jätta, et mitte spekuleerida kuni kõik on selge. Tabelisse 7 valiti välja tiheduste 10% - 50% tulemustest üks rida, selleks et näidata trendi läbi erinevate tiheduste. Graafikutena on eraldi välja toodud tulemused tihedustel 10% (Joonis 14), 40% (Joonis 15) ja 50% (Joonis 16) ning tabelitena: Tabel 8 (tihedus 10%), Tabel 9 (tihedus 40%) ja Tabel 10 (tihedus 50%), need on esitatud samuti trendi näitamiseks. Kõik ülejäänud erinevatel tihedustel saadud tulemused on esitatud graafikutena: Joonis 17 (tihedus 20%), Joonis 18 (tihedus 30%) ning tabelitena: Tabel 17 (tihedus 20%), Tabel 18 (tihedus 30%).

Tabelis 7 on esitatud kuue individuaalse algoritmi aegade tulemused ja veerus „Paralleelselt tööle pandud algoritm“ on välja toodud selle paralleelse algoritmi aeg, mis leidis graafis esimesena suurima kliki. Rohelise värviga on märgitud kõige kiirema algoritmi aeg, näiteks graafil, mille tihedus oli 0.1 ja tippude arv 2750 leidis kõige kiiremini lahenduse algoritm Largest-First V3 (LF V3), saades tulemuseks 1655 ms. Halli värviga on märgitud need paralleelselt tööle pandud algoritmide tulemused, mis jäid kiiruselt teisele või kolmandale kohale. Tihedusel 0.1 leidis paralleelselt tööle pandud algoritm graafil, mille tippude arv oli 2750 suurima kliki ajaga 2127 ms, mis on ajaliselt kolmas.

Tabel 7 Juhuslikel graafidel saadud tulemused.

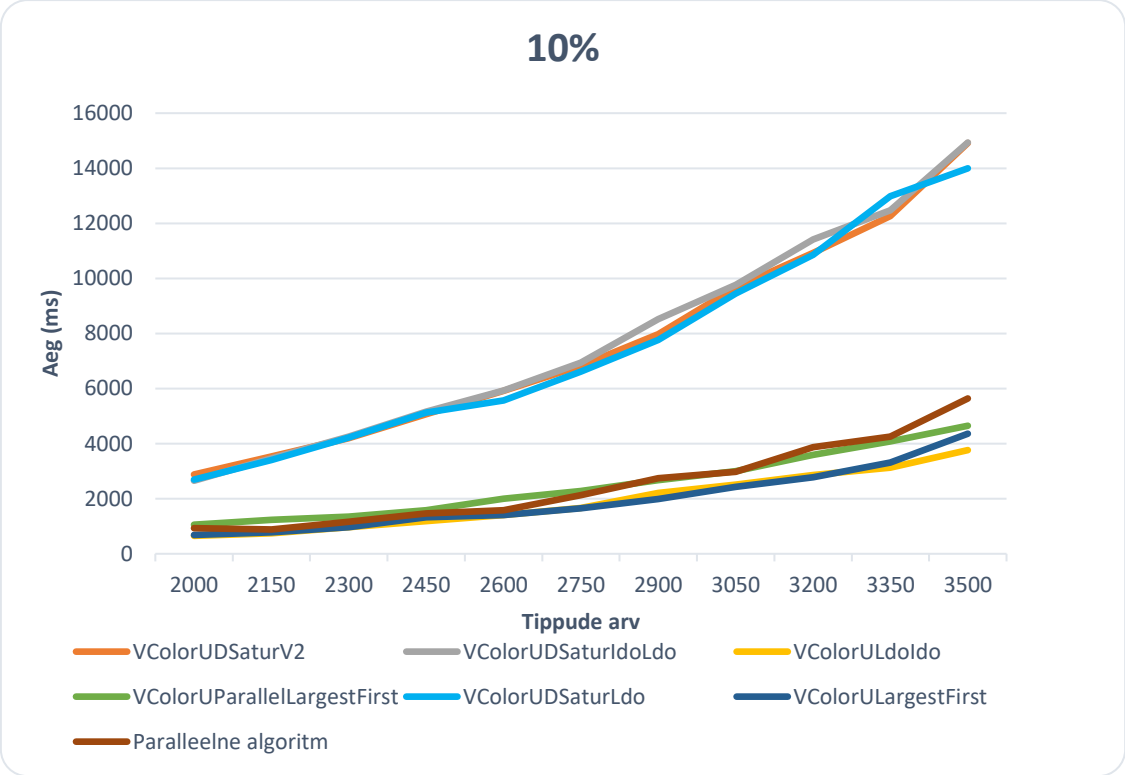
		Aeg (ms)						
Tihedus	**Tippude arv**	**Paralleelselt tööle pandud algoritm**	**DSatur V2**	**DSatur-IDO-LDO**	**LDO-IDO**	**PLF**	**DSatur-LDO**	**LF V3**
0.1	2750	2127	6819	6944	1660	2283	6619	1655
0.2	1150	1265	2022	2022	1060	1272	2104	1055
0.3	800	3552	3249	3221	2730	2970	3350	2666
0.4	500	3771	2769	2718	2893	2651	3029	2533
0.5	400	12165	7812	8756	8759	7373	8977	8758

Kõige kiirem algortim

Paralleelne algoritm jäi kiiruselt II või III kohale

Tabeli 7 tulemustest selgub, et paralleelselt tööle pandud algoritm ei leidnud ühelgi juhuslikul graafil kõige kiiremini tulemust võrreldes individuaalsete algoritmidega. Paralleelselt töötav algoritm jäi kiiruselt kolmandale kohale kahel graafil: tihedustel 0.1 ja 0.2. Viimasele kohale jäi paralleelne algoritm kolmel erineva tihedusega graafil: 0.3, 0.4, 0.5. Paralleelse algoritmi halb tulemus võis sõltuda sellest, et arvutil polnud piisavalt ressursse kuue algoritmi korraga jooksutamiseks või paralleliseerimine pidurdas protsessi.

Joonistel 14, 15 ja 16 on kujutatud graafikuna kõigi 11 testi tulemused tihedustel 10%, 40% ja 50% ja tabelites 8, 9 ja 10 on kujutatud samad tulemused samadel tihedustel numbritega. Graafikutes on algoritmidele määratud värvid, mis on kirjeldatud joonisel legendis.



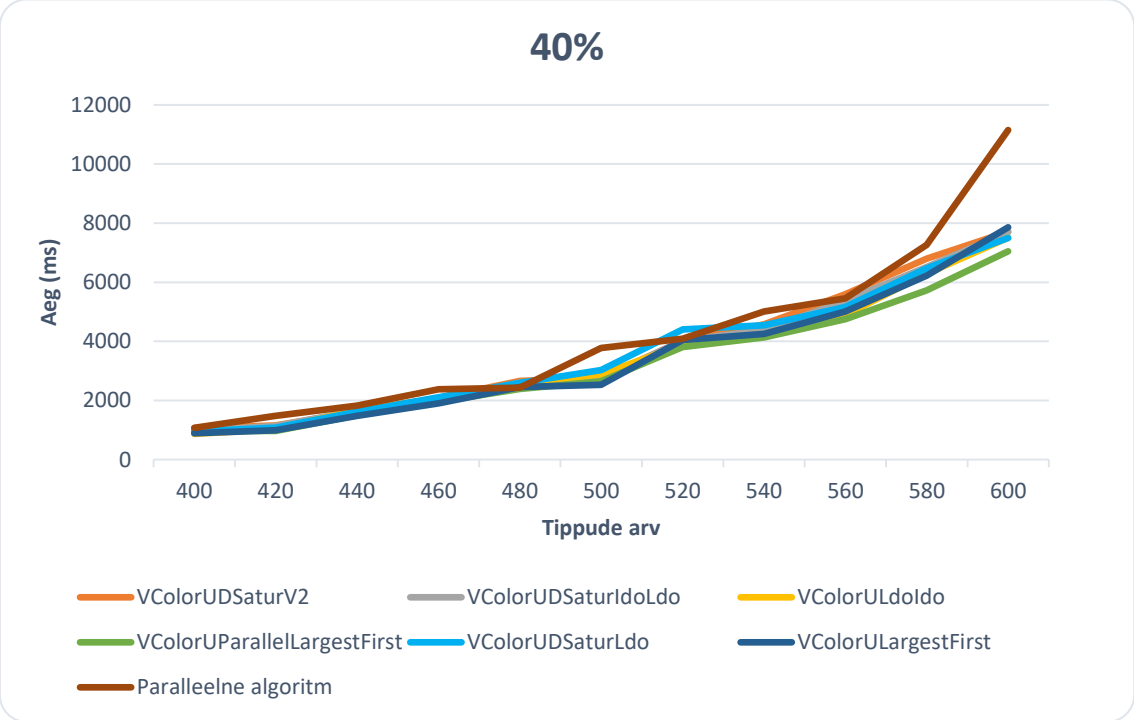
Joonis 14 Juhuslikult genereeritud graafidel saadud aegade tulemused esitatud graafikuna Tihedus 10%.

Tabel 8 Juhuslikult genereeritud graafidel saadud aegade tulemused esitatud tabelina. Tihedus 10%.

Tihedus 10%	**Aeg (ms)**						
Tippude arv	**Paralleelselt tööle pandud algoritm**	**DSatur V2**	**DSatur-IDO-LDO**	**LDO-IDO**	**PLF**	**DSatur-LDO**	**LF V3**
2000	935	2879	2653	652	1057	2694	684
2150	883	3538	3462	747	1232	3401	778
2300	1160	4196	4256	964	1348	4217	971
2450	1463	5068	5158	1192	1585	5127	1320
2600	1584	5920	5930	1410	1998	5566	1406
2750	2127	6819	6944	1660	2283	6619	1655

Tihedus 10%	Aeg (ms)						
Tippude arv	**Paralleelselt tööle pandud algoritm**	**DSatur V2**	**DSatur-IDO-LDO**	**LDO-IDO**	**PLF**	**DSatur-LDO**	**LF V3**
2900	2741	7974	8515	2206	2679	7762	1988
3050	2982	9671	9762	2514	3001	9454	2435
3200	3871	10926	11414	2858	3591	10856	2778
3350	4257	12266	12475	3122	4075	12979	3308
3500	5640	14915	14934	3765	4649	13997	4360

Joonisel 14 on näha, et algoritmide DSatur-IDO-LDO, DSatur V2 ja DSatur-LDO aegade tulemused on palju aeglasemad, kui ülejäänud algoritmidel. Paralleelse algoritmi tulemused hakkavad silma paistma suuremate tippude arvude puhul, alates tippude arvust 3050 kuni lõpuni on paralleelse algoritmi tumepunane joon kõrgemal kui algoritmide LDO-IDO, Largest-First ja Parallel Largest-First jooned.

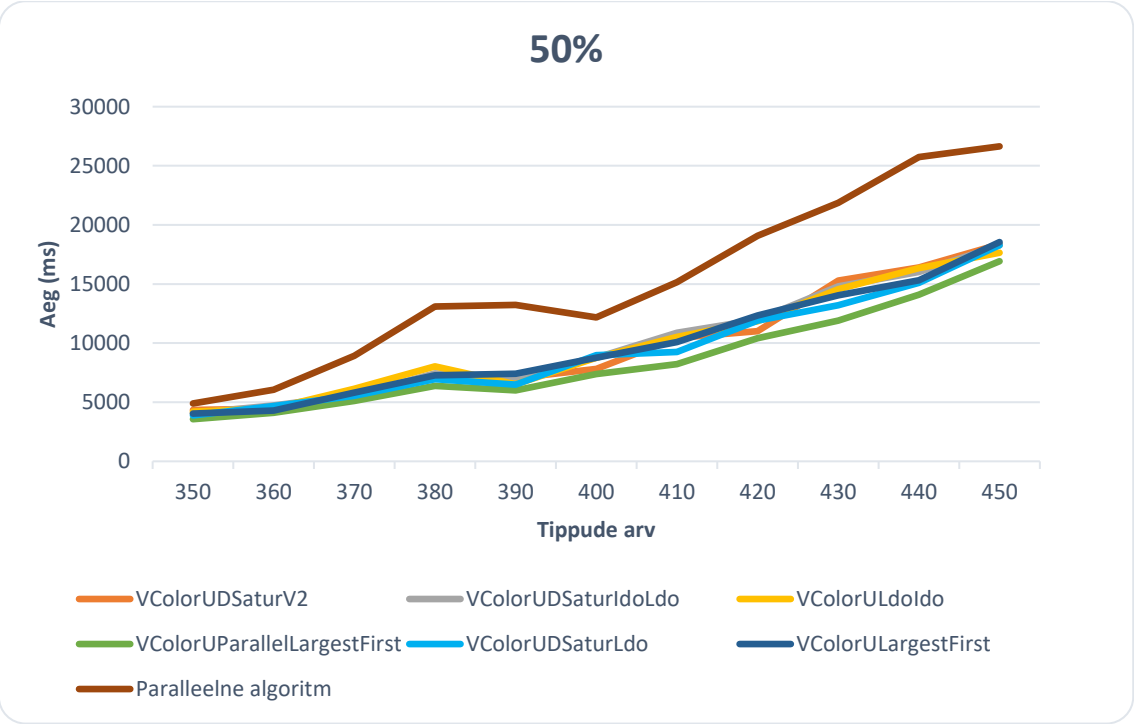


Joonis 15 Juhuslikult genereeritud graafidel saadud aegade tulemused esitatud graafikuna. Tihedus 40%.

Tabel 9 Juhuslikult genereeritud graafidel saadud aegade tulemused esitatud tabelina. Tihedus 40%.

Tihedus 40%	**Aeg (ms)**						
Tippude arv	**Paralleelselt tööle pandud algoritm**	**DSatur V2**	**DSatur-IDO-LDO**	**LDO-IDO**	**PLF**	**DSatur-LDO**	**LF V3**
400	1075	1068	987	870	976	953	891
420	1484	1155	1130	1028	970	1081	989
440	1824	1616	1666	1685	1507	1623	1480
460	2374	2085	2039	2064	1956	2104	1905
480	2435	2663	2609	2478	2394	2598	2459
500	3771	2769	2718	2893	2651	3029	2533
520	4095	4012	4034	3882	3805	4402	4043
540	5006	4579	4367	4192	4136	4544	4257
560	5453	5594	5382	4891	4750	5167	5007
580	7261	6800	6519	6285	5727	6477	6219
600	11150	7712	7695	7505	7048	7499	7859

Joonisel 15 eristub paralleelse algoritmi tumepunane joon üsna selgelt teistest värvilistest joontest. Paralleelne algoritm on peaaegu igal tippude arvul aeglasem kui teised. Graafidel, mille tippude arvud on 480, 520 ja 560 leidub mõni teine algoritm, mis on natukene aeglasem paralleelsest algoritmist, aga üldiselt paralleelne algoritm häid tulemusi ei näita sellel tihedusel.



Joonis 16 Juhuslikult genereeritud graafidel saadud aegade tulemused esitatud graafikuna Tihedus 50%.

Tabel 10 Juhuslikult genereeritud graafidel saadud aegade tulemused esitatud tabelina. Tihedus 50%.

Tihedus 50%	**Aeg (ms)**						
Tippude arv	**Paralleelselt tööle pandud algoritm**	**DSatur V2**	**DSatur-IDO-LDO**	**LDO-IDO**	**PLF**	**DSatur-LDO**	**LF V3**
350	4895	4347	4068	4201	3565	3886	4040
360	6045	4447	4733	4490	4096	4668	4287
370	8928	5824	5586	6124	5117	5538	5808
380	13109	7001	7427	8042	6391	6942	7269
390	13233	7044	7089	6511	6008	6502	7405
400	12165	7812	8756	8759	7373	8977	8758
410	15145	10429	10871	10533	8221	9240	10099
420	19078	11016	11978	11847	10406	11907	12331
430	21861	15286	14808	14560	11892	13196	14038
440	25738	16389	16018	16356	14105	15097	15316
450	26643	18362	18265	17645	16928	18287	18548

Joonisel 16 selgub, et juba tihedusel 50% hakkavad tekkima ebaproportsionaalsed tulemused, mida tuleb uurida edaspidi nii nagu on kirjeldatud peatükis 4.3.1 Edaspidised uuringud. Paralleelne algoritm lahendas kõiki graafe kõige aeglasemalt. Tippude arvude suurenedes läheb paralleelse algoritmi aegade vahe võrreldes teiste algoritmidega järjest suuremaks.

4.3 Järeldused

DIMACS graafidel andsid paralleliseeritud algoritmid umbes pooltel graafidel paremaid ajalisi tulemusi kui enamus individuaalselt töötavaid algoritme. Paralleelselt tööle pandud algoritm jäi esikolmikusse oma aja tulemusega 13 korral. Küsimusi tekitab see, et miks paralleelne algoritm ei andnud võrdset tulemust sama algoritmi individuaalse tulemusega. Näiteks graafil p_hat300-2.clq leidis Largest-First V3 paralleelse algoritmi alamosana kõige kiiremini lahenduse ajaga 1066 ms, aga sama algoritmi individuaalne käivitamine andis sellel graafil tulemuseks 619 ms. Teoreetiliselt peaks sama algoritm andma sarnase tulemuse paralleelsel ja individuaalsel käivitamisel, aga sellel näitel on vahe liiga suur.

Analüüsitud harude arvu tulemuste põhjal jäi paralleelne algoritm esikolmikusse 24 graafil oma tulemusega. Selle põhjal võiks järeldada, et harude analüüsimine ei mõjutanud paralleelse algoritmi lahendamise aega, sest analüüsitud harude arv oli 18 graafil kõige väiksem, kuuel graafil jäi paralleelse algoritmi tulemus teisele või kolmandale kohale. Kahel graafil jäi paralleelne algoritm oma tulemusega viimasele kohale, aga mõlemal korral leidus veel üks algoritm täpselt sama tulemusega, paralleelne algoritm ja individuaalne algoritm olid samad mõlemal juhul. Võrredes graafidel saadud analüüsitud harude tulemusi, on näha, et paljudel graafidel paralleelse algoritmi tulemus ja individuaalselt töötava algoritmi tulemused on samad. Ei esine selliseid juhtumeid nagu aegade tulemuses näitena välja toodi, seega paralleelse algoritmi puhul mõjutab lahendamise aega mingi muu tegur.

Leitud suurima kliki tulemusi ja kasutatud värvide arvu tulemusi võrreldes ei esine samuti suuri erinevusi paralleelsete algoritmide ja individuaalselt töötavate algoritmide tulemustes.

Juhuslikel graafidel saadud tulemustes ei näidanud paralleelsed algoritmid häid tulemusi, suurtel tihedustel tekkisid ebaproportsionaalsed tulemused, adekvaatsete tulemuste saamiseks oleks pidanud juhuslikel graafidel teostama alamteste rohkem kui 1 ja võtma tulemustesse nende alamtestide tulemustest keskmise.

Paralleelsete algoritmide ebaõnnestumine võib olla põhjustatud mitmest asjast:

- Paralleelselt töötavad algoritmid nõuavad palju ressurssi arvutilt, neljatuumaline protsessor võis jääda lahjaks sellele programmile.
- Liiga palju algoritme toimetas paralleelselt, vähemate algoritmide paraleelselt tööle panemine oleks võib-olla paremaid tulemusi andnud.
- Paralleliseerimine ise pidurdas kogu protsessi liiga palju.
- Valitud algoritmid polnud kõige sobivamad, valimata jäänud algoritmid oleksid äkki paremini toime tulnud.

4.3.1 Edaspidised uuringud

Selle töö käigus tekkisid küsimused, mis jäid selle uurimuse käigus lahendamata, aga millesse tasuks süveneda, et saada paremaid tulemusi suurima kliki leidmisel.

Kõigepealt tuleks teostada rohkem alamteste juhuslikel graafidel, mille tihedus on suurem kui 50%, et vältida ebaproportsionaalsete tulemuste saamist.

Järgmine asi, mida tasub uurida, on see, et miks DIMACS graafidel paralleelselt tööle pandud algoritmid ei anna samaväärseid aja tulemusi individuaalselt tööle pandud algoritmidega. Kuna paralleliseerimine koormab arvutit päris palju, tuleb proovida sama programmi käivitamist paremate omadustega arvutil ja tulemusi võib parandada ka algoritmide arvu vähendamine paralleelses programmis. Tabel 11 näitab, et mõndade graafide puhul jõuab üks algoritm enne tulemuseni, kui mõni teine algoritm jõuab üldse alustada oma tööd, sellest võib järeldada, et ressursse jäi arvutil väheks.

Veel tasub eksperimenteerida muude heuristiliste värvimisalgoritmidega suurima kliki leidmise algoritmi paralleliseerimist ja uurida teisi paralleliseerimise võimalusi.

5 Kokkuvõte

Töö peamine eesmärk oli uurida paralleliseerimise mõju suurima kliki leidmise algoritmile, mis sisaldab erinevaid heuristilisi värvimise tehnikaid.

Selle töö käigus uuriti heuristilisi värvimisalgoritme ja nende eripärasid ning valiti välja kuus värvimisalgoritmi, mida kombineeriti suurima kliki leidmise algoritmiga VColor-u. Konstrueeriti metaalgoritm suurima kliki leidmise algoritmi paralleliseerimiseks ja teostati katseid juhuslikel ja DIMACS graafidel. Algoritmide saadud tulemusi võrreldi omavahel ja selgus, et paralleliseerimine ei andnud nii häid tulemusi, kui algselt oodati.

Paralleliseerimine ei mõjutanud kuidagi harude analüüsimist, suurimat klikki ja värvide kasutamist DIMACS graafidel, ainult ajaline tulemus oli paralleelselt tööle pandud algoritmidel mitmel graafil kõrgem kui individuaalselt tööle pandud algoritmidel. Aja tulemus võis sõltuda sellest, et paralleelselt pandi tööle liiga palju algoritme ja arvuti ressurssidest ei piisanud, et korraga nii palju algoritme lahendust saaks otsida. Paralleliseerimise protsess ise võis samuti aega mõjutada, kuna igale algoritmile loodi eraldi *Task* ja aega kulus ka algoritmide töö katkestamise peale, kui üks algoritm oli lahenduse leidnud.

Juhuslike graafide puhul tekkisid suurematel tihedustel ebaproportsionaalsed tulemused, mida selles töös ei analüüsitud. Väiksematel graafi tihedustel paralleelne algoritm ei andnud häid tulemusi, iga tiheduse puhul tasub teha mitu alamtesti ja tulemustes esitada nende alamtestide keskmine tulemus, et saada paremat pilti paralleelse algoritmi üldisest jõudlusest.

Kasutatud kirjandus

[1] D. Kumlander, Some Practical Algorithms to Solve The Maximum Clique Problem, Tallinn, 2005.

[2] M. Koit, „Graafiteooria põhimõisted," [Võrgumaterjal]. Available: http://www.teaduskool.ut.ee/sites/default/files/teaduskool/oppetoo/mat_gymn_graafiteooria.pdf. [Kasutatud 21. 05. 2017].

[3] A. Buldas, P. Laud ja J. Villemson, Graafid, Tartu, 2003.

[4] I. Petuhhov, „Graaf," [Võrgumaterjal]. Available: http://www.cs.tlu.ee/~inga/alg_andm_09/graph_2008.pdf. [Kasutatud 21.05.2017].

[5] H. Nestra, „Graafid," [Võrgumaterjal]. Available: http://kodu.ut.ee/~nestra/mat/inf/p/aa/08s/s/graafid.pdf. [Kasutatud 22.05.2017].

[6] A. Kulitškov, Investigating effects of applying different heuristic coloring techniques on modern maximum clique algorithms, Tallinn, 2016.

[7] „Induced subgraph," Wikimedia Foundation, Inc., [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/Induced_subgraph. [Kasutatud 21.05.2017].

[8] M. Kubale, Graph colorings, 2004.

[9] „Sissejuhatus teoreetilisse informaatikasse," 2015. [Võrgumaterjal]. Available: https://courses.cs.ut.ee/MTAT.05.125/2015_fall/uploads/Main/SolFinal-fall2015-14jan-EST.pdf. [Kasutatud 21.05.2017].

[10] J. Penjam, „Keerukusteooria elemente," [Võrgumaterjal]. Available: http://www.cs.ioc.ee/RekKursus/Handouts/10_Keerukus.pdf. [Kasutatud 21.05.2017].

[11] A. Porošin, Reversed Search Maximum Clique Algorithm Based on Recoloring, Tallinn, 2015.

[12] W. Hasenplaugh, T. Kaler, T. B. Schardl ja C. E. Leiserson, Ordering Heuristics for Parallel Graph Coloring, 2014.

[13] Soma Saha, Gyan Baboo, Rajeev Kumar, An Efficient EA with Multipoint Guided Crossover for Bi-objective Graph Coloring Problem, 2011.

[14] J. R. Allwright, R. Bordawekar, P. D. Coddington, K. Dincer ja C. L. Martin, A Comparison of Parallel Graph Coloring Algorithms, 1995.

[15] D. S. Johnson ja M. A. Trick, Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, October 11-13, 1993, 1996.

Lisa 1 – DIMACS graafid – Paralleelselt tööle pandud algoritmide tulemused

Tabel 11 DIMACS graafidel paralleelselt tööle pandud algoritmide tulemused – aeg.

Graaf	Aeg (ms)					
	DSatur V2	**DSatur-IDO-LDO**	**LDO-IDO**	**PLF**	**DSatur-LDO**	**LF V3**
c-fat500-1.clq	77 (c)	83 (c)	24	75 (c)	46 (c)	(c)
c-fat500-2.clq	49 (c)	73 (c)	7	135 (c)	64 (c)	(c)
c-fat500-5.clq	108 (c)	100 (c)	21	173 (c)	91 (c)	(c)
c-fat500-10.clq	174 (c)	288 (c)	26	330 (c)	171 (c)	(c)
hamming10-2.clq	6776 (c)	6784 (c)	298	2144 (c)	6497 (c)	(c)
hamming6-2.clq	4	(c)	0 (c)	21 (c)	(c)	(c)
hamming6-4.clq	1 (c)	2	0 (c)	(c)	(c)	(c)
hamming8-2.clq	116	116 (c)	13 (c)	(c)	(c)	(c)
hamming8-4.clq	173 (c)	107	5 (c)	(c)	(c)	(c)
johnson16-2-4.clq	451 (c)	252	1 (c)	(c)	(c)	(c)
johnson8-2-4.clq	0 (c)	20	2 (c)	(c)	(c)	(c)
johnson8-4-4.clq	2	2 (c)	(c)	(c)	(c)	(c)
keller4.clq	450	23 (c)	(c)	(c)	(c)	(c)
MANN_a27.clq	130586(c)	127985(c)	128117(c)	126534	123821(c)	118615(c)
MANN_a9.clq	5	2 (c)	10 (c)	203 (c)	3 (c)	3 (c)
p_hat300-1.clq	70 (c)	53 (c)	43 (c)	161 (c)	54	38 (c)
p_hat300-2.clq	1188 (c)	1191 (c)	1191 (c)	1201 (c)	1225 (c)	1066
p_hat300-3.clq	1145614 (c)	1145630 (c)	1145636 (c)	1145492	1145748 (c)	1145751 (c)
p_hat500-1.clq	494	651 (c)	320 (c)	903 (c)	604 (c)	307 (c)
p_hat500-2.clq	175595(c)	175711(c)	175833(c)	175518	175843(c)	175677(c)
p_hat700-1.clq	1241 (c)	1453 (c)	1507 (c)	1643 (c)	1636 (c)	1150

Graaf	Aeg (ms)					
	DSatur V2	**DSatur-IDO-LDO**	**LDO-IDO**	**PLF**	**DSatur-LDO**	**LF V3**
p_hat1000-1.clq	7949 (c)	8080 (c)	8020 (c)	8126 (c)	8098 (c)	7785
san1000.clq	19444(c)	19528(c)	19571(c)	19321	19632(c)	19660(c)
san200_0.7_1.clq	706 (c)	432	913 (c)	973 (c)	980 (c)	967 (c)
san200_0.7_2.clq	78 (c)	135 (c)	121 (c)	307 (c)	151 (c)	116
san200_0.9_1.clq	1319 (c)	1345 (c)	1193	1324 (c)	1340 (c)	1407 (c)
san400_0.5_1.clq	1355 (c)	1196	1337 (c)	1395 (c)	1393 (c)	1412 (c)

(c) Algoritmi töö katkestati

Algortim, mis leidis esimesena lahenduse

Tabel 12 DIMACS graafidel paralleelselt tööle pandud algoritmide tulemused – suurim klikk.

Graaf	Suurim klikk	Suurim klikk					
		Dsatur V2	**DSatur-IDO-LDO**	**LDO-IDO**	**PLF**	**Dsatur-LDO**	**LF V3**
c-fat500-1.clq	14	0 (c)	0 (c)	14	0 (c)	0 (c)	(c)
c-fat500-2.clq	26	0 (c)	0 (c)	26	0 (c)	0 (c)	(c)
c-fat500-5.clq	64	0 (c)	0 (c)	64	0 (c)	0 (c)	(c)
c-fat500-10.clq	126	0 (c)	0 (c)	126	0 (c)	0 (c)	(c)
hamming10-2.clq	512	0 (c)	0 (c)	512	0 (c)	0 (c)	(c)
hamming6-2.clq	32	32	(c)	0 (c)	0 (c)	(c)	(c)
hamming6-4.clq	4	4 (c)	4	0 (c)	(c)	(c)	(c)
hamming8-2.clq	128	128	128 (c)	0 (c)	(c)	(c)	(c)
hamming8-4.clq	16	0 (c)	16	0 (c)	(c)	(c)	(c)
johnson16-2-4.clq	8	0 (c)	8	0 (c)	(c)	(c)	(c)
johnson8-2-4.clq	4	4 (c)	4	0 (c)	(c)	(c)	(c)
johnson8-4-4.clq	14	14	14 (c)	14 (c)	(c)	(c)	(c)
keller4.clq	11	11	0 (c)	(c)	(c)	(c)	(c)
MANN_a27.clq	126	0 (c)	0 (c)	0 (c)	126	0 (c)	0 (c)
MANN_a9.clq	16	16	16 (c)	16 (c)	0 (c)	16 (c)	16 (c)

Graaf	Suurim klikk	Suurim klikk					
		Dsatur V2	**DSatur-IDO-LDO**	**LDO-IDO**	**PLF**	**Dsatur-LDO**	**LF V3**
p_hat300-1.clq	8	8 (c)	8 (c)	8 (c)	0 (c)	8	8 (c)
p_hat300-2.clq	25	0 (c)	0 (c)	0 (c)	0 (c)	0 (c)	25
p_hat300-3.clq	36	0 (c)	0 (c)	0 (c)	36	0 (c)	0 (c)
p_hat500-1.clq	9	9	9 (c)	9 (c)	0 (c)	9 (c)	9 (c)
p_hat500-2.clq	36	0 (c)	0 (c)	0 (c)	36	0 (c)	0 (c)
p_hat700-1.clq	11	0 (c)	0 (c)	0 (c)	0 (c)	0 (c)	11
p_hat1000-1.clq	10	0 (c)	0 (c)	10 (c)	0 (c)	0 (c)	10
san1000.clq	15	0 (c)	0 (c)	0 (c)	15	0 (c)	0 (c)
san200_0.7_1.clq	30	0 (c)	30	0 (c)	0 (c)	0 (c)	0 (c)
san200_0.7_2.clq	18	18 (c)	18 (c)	18 (c)	0 (c)	18 (c)	18
san200_0.9_1.clq	70	0 (c)	0 (c)	70	0 (c)	0 (c)	0 (c)
san400_0.5_1.clq	13	0	13	0 (c)	0 (c)	0 (c)	0 (c)

(c) Algoritmi töö katkestati

Algortim, mis leidis esimesena lahenduse

Tabel 13 DIMACS graafidel paralleelselt tööle pandud algoritmide tulemused – kasutatud värvide arv.

Graaf	Kasutatud värvide arv					
	Dsatur V2	**DSatur-IDO-LDO**	**LDO-IDO**	**PLF**	**DSatur-LDO**	**LF V3**
c-fat500-1.clq	14 (c)	14 (c)	14	17 (c)	(c)	(c)
c-fat500-2.clq	26 (c)	26 (c)	26	31 (c)	(c)	(c)
c-fat500-5.clq	64 (c)	64 (c)	64	73 (c)	64 (c)	(c)
c-fat500-10.clq	126 (c)	126 (c)	126	126 (c)	126 (c)	126 (c)
hamming10-2.clq	512 (c)	512 (c)	512	540 (c)	512 (c)	512 (c)
hamming6-2.clq	32	(c)	(c)	(c)	(c)	(c)
hamming6-4.clq	8 (c)	7	(c)	(c)	(c)	(c)
hamming8-2.clq	128	128 (c)	(c)	(c)	(c)	(c)
hamming8-4.clq	24 (c)	16	32 (c)	(c)	(c)	(c)
johnson16-2-4.clq	17 (c)	14	14 (c)	15 (c)	14 (c)	14 (c)

Graaf	Kasutatud värvide arv					
	Dsatur V2	**DSatur-IDO-LDO**	**LDO-IDO**	**PLF**	**DSatur-LDO**	**LF V3**
johnson8-2-4.clq	6 (c)	6	6 (c)	6 (c)	6 (c)	6 (c)
johnson8-4-4.clq	17	14 (c)	20 (c)	21 (c)	17 (c)	20 (c)
keller4.clq	23	25 (c)	39 (c)	27 (c)	24 (c)	39 (c)
MANN_a27.clq	136 (c)	140 (c)	0 (c)	144	140 (c)	141 (c)
MANN_a9.clq	19	19 (c)	16 (c)	21 (c)	19 (c)	(c)
p_hat300-1.clq	21 (c)	22 (c)	8 (c)	24 (c)	22	(c)
p_hat300-2.clq	43 (c)	42 (c)	25 (c)	45 (c)	42 (c)	43
p_hat300-3.clq	70 (c)	70 (c)	35 (c)	73	69 (c)	71 (c)
p_hat500-1.clq	30	32 (c)	9 (c)	34 (c)	32 (c)	33 (c)
p_hat500-2.clq	65 (c)	66 (c)	68 (c)	68	66 (c)	68 (c)
p_hat700-1.clq	40 (c)	41 (c)	41 (c)	44 (c)	40 (c)	41
p_hat1000-1.clq	53 (c)	52 (c)	54 (c)	57 (c)	52 (c)	55
san1000.clq	26 (c)	26 (c)	28 (c)	15	24 (c)	29 (c)
san200_0.7_1.clq	38 (c)	35	43 (c)	42 (c)	42 (c)	45 (c)
san200_0.7_2.clq	20 (c)	23 (c)	22 (c)	18 (c)	23 (c)	24
san200_0.9_1.clq	76 (c)	70 (c)	77	75 (c)	73 (c)	78 (c)
san400_0.5_1.clq	19 (c)	20	22 (c)	13 (c)	21 (c)	23 (c)

(c)	Algoritmi töö katkestati
	Algortim, mis leidis esimesena lahenduse

Tabel 14 DIMACS graafidel paralleelselt tööle pandud algoritmide tulemused – analüüsitud harude arv.

Graaf	**Analüüsitud harude arv**					
	Dsatur V2	**DSatur-IDO-LDO**	**LDO-IDO**	**PLF**	**DSatur-LDO**	**LF V3**
c-fat500-1.clq	0 (c)	0 (c)	14	0 (c)	0 (c)	(c)
c-fat500-2.clq	0 (c)	0 (c)	26	0 (c)	0 (c)	(c)
c-fat500-5.clq	0 (c)	0 (c)	64	0 (c)	0 (c)	(c)
c-fat500-10.clq	0 (c)	0 (c)	126	0 (c)	0 (c)	(c)
hamming10-2.clq	0 (c)	0 (c)	512	0 (c)	0 (c)	(c)
hamming6-2.clq	32	(c)	0 (c)	0 (c)	(c)	(c)
hamming6-4.clq	340 (c)	352	0 (c)	(c)	(c)	(c)
hamming8-2.clq	128	128 (c)	0 (c)	(c)	(c)	(c)
hamming8-4.clq	34342 (c)	215	0 (c)	(c)	(c)	(c)
johnson16-2-4.clq	608159 (c)	353522	0 (c)	(c)	(c)	(c)
johnson8-2-4.clq	38 (c)	37	364 (c)	(c)	(c)	(c)
johnson8-4-4.clq	364	18 (c)	(c)	(c)	(c)	(c)
keller4.clq	435550	0 (c)	(c)	(c)	(c)	(c)
MANN_a27.clq	27680987 (c)	8925567 (c)	19274352 (c)	8110992	8772194 (c)	20954191 (c)
MANN_a9.clq	5224	2171 (c)	4093 (c)	0 (c)	2171 (c)	4156 (c)
p_hat300-1.clq	25170 (c)	26001 (c)	23352 (c)	0 (c)	25193	23068 (c)
p_hat300-2.clq	548276(c)	517525 (c)	403232(c)	444525(c)	476676(c)	394533
p_hat300-3.clq	378743067 (c)	3869100 05 (c)	37169784 3 (c)	38580465 8	376265796 (c)	37964910 2 (c)
p_hat500-1.clq	268503	278488 (c)	254312(c)	0(c)	272261(c)	261465(c)
p_hat500-2.clq	62238832 (c)	5905311 5 (c)	48244381 (c)	45868870	53735668 (c)	50812293 (c)
p_hat700-1.clq	484039(c)	377757 (c)	731202(c)	240023(c)	791351(c)	977854
p_hat1000-1.clq	5110042 (c)	5559320 (c)	6539346 (c)	5980502 (c)	5729271 (c)	6556251

Graaf	Analüüsitud harude arv					
	Dsatur V2	**DSatur-IDO-LDO**	**LDO-IDO**	**PLF**	**DSatur-LDO**	**LF V3**
san1000.clq	813788 (c)	1078619 (c)	3421716 (c)	1329301	10190899 (c)	3332248 (c)
san200_0.7_1.clq	777103 (c)	87844	1010448 (c)	1891066 (c)	1196090 (c)	223928 (c)
san200_0.7_2.clq	3502 (c)	18488 (c)	23459 (c)	0 (c)	9596 (c)	35546
san200_0.9_1.clq	795196 (c)	1616111 (c)	995610	1104201 (c)	255379(c)	432085(c)
san400_0.5_1.clq	378639(c)	267929	302584(c)	637237(c)	1777135 (c)	903835(c)

(c) Algoritmi töö katkestati

Algortim, mis leidis esimesena lahenduse

Lisa 2 - DIMACS graafid – Järjestikku tööle pandud algoritmide tulemused

Tabel 15 DIMACS graafidel järjestikku tööle pandud algoritmide tulemused – suurim klikk.

Graaf	Suurim klikk	Suurim klikk					
		DSatur V2	DSatur-IDO-LDO	LDO-IDO	PLF	DSatur-LDO	LF V3
c-fat500-1.clq	14	14	14	14	14	14	14
c-fat500-2.clq	26	26	26	26	26	26	26
c-fat500-5.clq	64	64	64	64	64	64	64
c-fat500-10.clq	126	126	126	126	126	126	126
hamming10-2.clq	512	512	512	512	261	512	512
hamming6-2.clq	32	32	32	32	32	32	32
hamming6-4.clq	4	4	4	4	4	4	4
hamming8-2.clq	128	128	128	128	67	128	128
hamming8-4.clq	16	16	16	16	16	16	16
johnson16-2-4.clq	8	8	8	8	8	8	8
johnson8-2-4.clq	4	4	4	4	4	4	4
johnson8-4-4.clq	14	14	14	14	14	14	14
keller4.clq	11	11	11	11	11	11	11
MANN_a27.clq	126	123	126	125	126	126	123
MANN_a9.clq	16	16	16	16	16	16	16
p_hat300-1.clq	8	8	8	8	8	8	8
p_hat300-2.clq	25	25	25	25	25	25	25
p_hat300-3.clq	36	34	34	36	34	35	35
p_hat500-1.clq	9	9	9	9	9	9	9
p_hat500-2.clq	36	36	36	36	36	36	36
p_hat700-1.clq	11	11	11	11	11	11	11
p_hat1000-1.clq	10	10	10	10	10	10	10

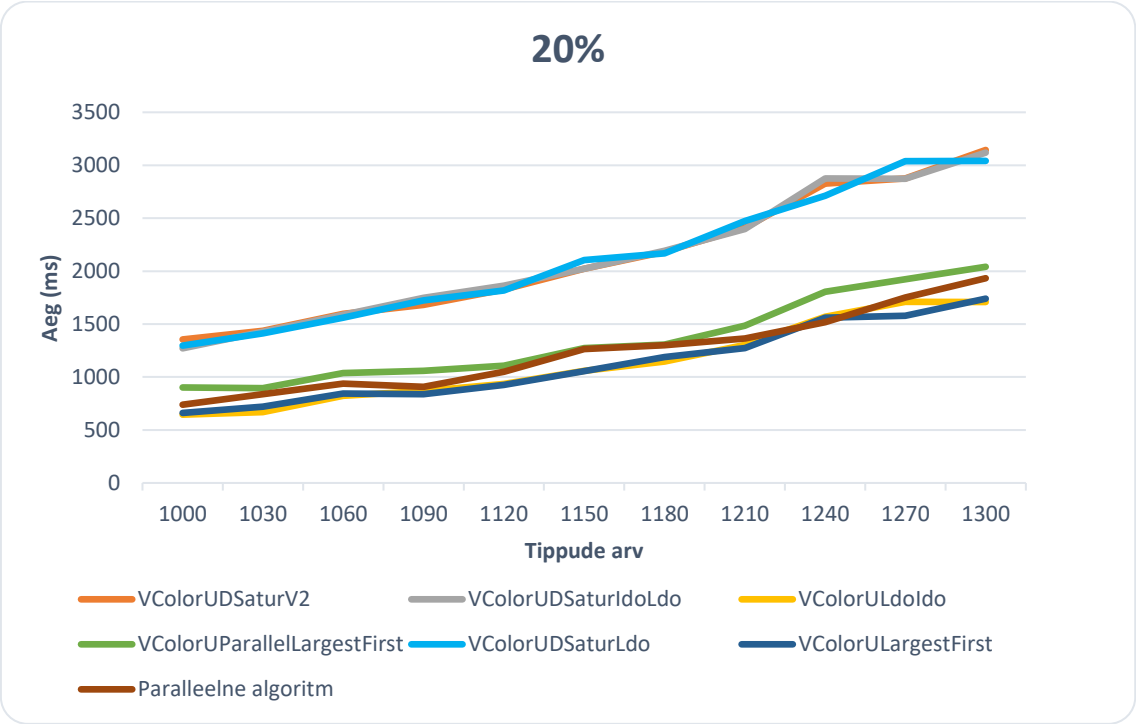
Graaf	Suurim klikk	Suurim klikk					
		DSatur V2	DSatur-IDO-LDO	LDO-IDO	PLF	DSatur-LDO	LF V3
san1000.clq	15	15	15	10	15	15	10
san200_0.7_1.clq	30	18	30	19	16	30	30
san200_0.7_2.clq	18	18	18	18	18	18	18
san200_0.9_1.clq	70	70	70	70	70	70	48
san400_0.5_1.clq	13	13	13	13	13	13	13

Tabel 16 DIMACS graafidel järjestikku tööle pandud algoritmide tulemused – kasutatud värvide arv.

Graaf	Kasutatud värvide arv					
	DSatur V2	DSatur-IDO-LDO	LDO-IDO	PLF	DSatur-LDO	LF V3
c-fat500-1.clq	14	14	14	18	14	14
c-fat500-2.clq	26	26	26	32	26	26
c-fat500-5.clq	64	64	64	73	64	64
c-fat500-10.clq	126	126	126	126	126	126
hamming10-2.clq	512	512	512	541	512	512
hamming6-2.clq	32	32	32	35	32	32
hamming6-4.clq	8	7	8	10	7	8
hamming8-2.clq	128	128	128	136	128	128
hamming8-4.clq	24	16	32	32	24	32
johnson16-2-4.clq	17	14	14	14	14	14
johnson8-2-4.clq	6	6	6	6	6	6
johnson8-4-4.clq	17	14	20	20	17	20
keller4.clq	23	25	39	29	24	39
MANN_a27.clq	136	140	142	144	140	141
MANN_a9.clq	19	19	20	21	19	19
p_hat300-1.clq	21	22	22	24	22	23
p_hat300-2.clq	43	42	44	46	42	43
p_hat300-3.clq	70	70	69	72	69	71

Graaf	Kasutatud värvide arv					
	DSatur V2	**DSatur-IDO-LDO**	**LDO-IDO**	**PLF**	**DSatur-LDO**	**LF V3**
p_hat500-1.clq	30	32	34	34	32	33
p_hat500-2.clq	65	66	68	68	66	68
p_hat700-1.clq	40	41	41	43	40	41
p_hat1000-1.clq	53	52	54	56	52	55
san1000.clq	26	26	28	15	24	29
san200_0.7_1.clq	38	35	43	43	42	45
san200_0.7_2.clq	20	23	22	18	23	24
san200_0.9_1.clq	76	70	77	76	73	78
san400_0.5_1.clq	19	20	22	13	21	23

Lisa 3 – Juhuslikel graafidel saadud tulemused tabelitena ja graafikutena – aeg

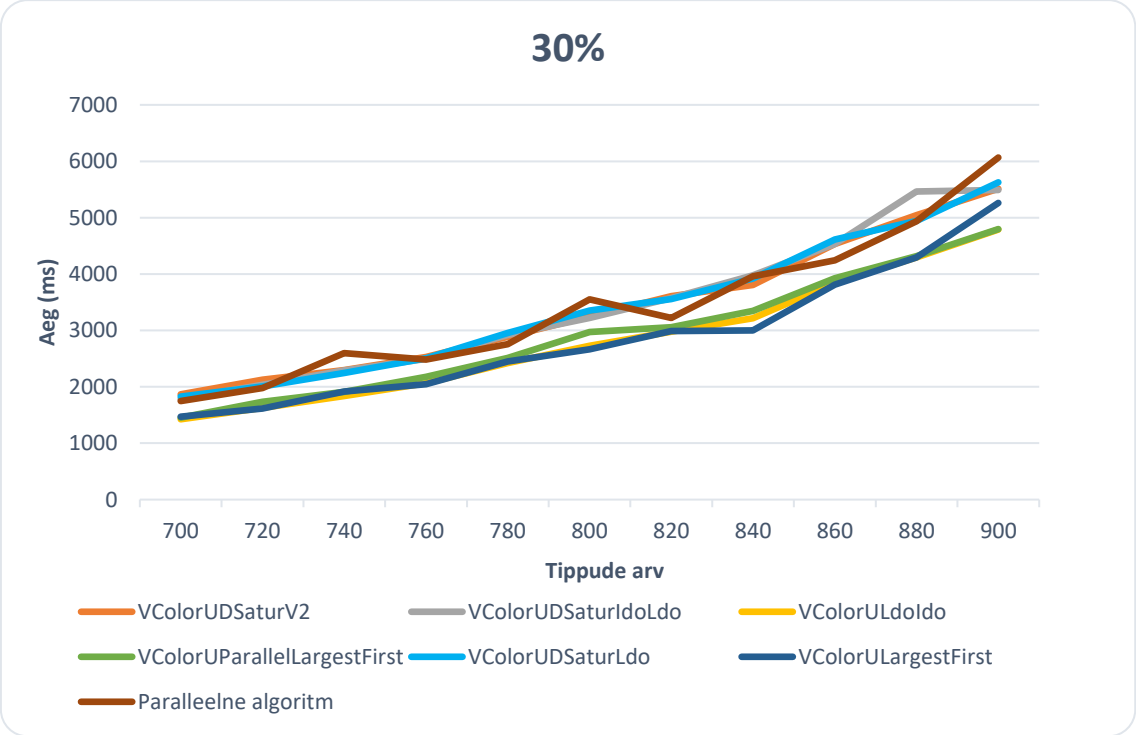


Joonis 17 Juhuslikult genereeritud graafidel saadud aegade tulemused esitatud graafikuna. Tihedus 20%.

Tabel 17 Juhuslikult genereeritud graafidel saadud aegade tulemused esitatud tabelina. Tihedus 20%.

Tihedus 20%	**Aeg (ms)**						
Tippude arv	**Paralleelselt tööle pandud algoritm**	**DSatur V2**	**DSatur-IDO-LDO**	**LDO-IDO**	**PLF**	**DSatur-LDO**	**LF V3**
1000	739	1355	1271	645	903	1297	661
1030	837	1437	1426	668	895	1412	721
1060	937	1597	1587	823	1037	1562	845
1090	908	1680	1749	868	1060	1723	838
1120	1050	1825	1862	936	1106	1816	927

Tihedus 20%	Aeg (ms)						
Tippude arv	**Paralleelselt tööle pandud algoritm**	**DSatur V2**	**DSatur-IDO-LDO**	**LDO-IDO**	**PLF**	**DSatur-LDO**	**LF V3**
1150	1265	2022	2022	1060	1272	2104	1055
1180	1300	2182	2191	1147	1306	2167	1189
1210	1365	2409	2397	1304	1484	2474	1274
1240	1517	2826	2875	1569	1804	2713	1561
1270	1750	2874	2872	1710	1923	3038	1580
1300	1933	3144	3119	1708	2041	3041	1740



Joonis 18 Juhuslikult genereeritud graafidel saadud aegade tulemused esitatud graafikuna. Tihedus 30%.

Tabel 18 Juhuslikult genereeritud graafidel saadud aegade tulemused esitatud tabelina. Tihedus 30%.

Tihedus 30%	**Aeg (ms)**						
Tippude arv	**Paralleelselt tööle pandud algoritm**	**DSatur V2**	**DSatur-IDO-LDO**	**LDO-IDO**	**PLF**	**DSatur-LDO**	**LF V3**
700	1748	1867	1783	1425	1454	1827	1471
720	1977	2126	2037	1629	1736	2009	1615
740	2597	2298	2291	1835	1912	2243	1918
760	2485	2530	2496	2063	2178	2506	2046
780	2759	2885	2922	2422	2514	2953	2448
800	3552	3249	3221	2730	2970	3350	2666
820	3224	3610	3572	2975	3058	3558	2990
840	3961	3806	3978	3215	3348	3920	3001
860	4244	4535	4532	3840	3927	4611	3811
880	4936	5046	5465	4292	4320	4944	4296
900	6066	5513	5491	4789	4797	5626	5261