TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Rain Haabu 163895IASB

SUDOKU LAHENDAJA OPTIMEERIMINE KIIRUSELE

Bakalaureusetöö

Juhendaja: Peeter Ellervee
PhD

Tallinn 2019

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: Rain Haabu

20.05.2019

Annotatsioon

Käesoleva töö esmaseks eesmärgiks oli realiseerida Java programmeerimiskeeles sudoku lahendaja koos graafilise kasutajaliidesega, mis on optimiseeritud kiirusele.

Töö käigus uuriti erinevaid lahendusmeetodeid sudoku lahendamiseks. Töös on kasutusel loogilistest lahendusmeetoditest üksikkandidaatide ning peidetud üksikkandidaatide lahendusmeetodid. Lisaks kasutatakse veel tagurdamise lahendusmeetodit.

Töös testiti erinevaid lahendusstrateegiad sudoku lahendamiseks kasutades töös olevaid lahendusmeetodeid. Testimise eesmärgiks oli leida kiire lahendusstrateegia sudoku lahendamiseks.

Töös disainiti ning realiseeriti programm sudoku lahendamiseks kasutades töös testimise tulemusel valminud kõige kiiremat lahendusstrateegiat. Programmis on realiseeritud ka graafiline kasutajaliides, millele on võimalik kasutajal sisestada uus sudoku ning sellel teha muudatusi. Kasutaja saab kasutajaliidesest sudoku ära lahendada, mille tagajärjel kuvatakse kasutajale lahendamisaeg ning loogiliste lahendusmeetodite sammud mis tehti sudoku lahendamisel. Lisaks saab kasutaja kasutajaliidesest küsida lahendatava sudoku kohta loogiliste lahendusmeetodite järgi vihjeid ning kontrollida, kas sudokul on konflikte sudoku väärtuste paigutusega.

Töö tulemusena arendati välja antud töö kontekstis uuritud lahendusmeetoditega optimaalne lahendusstrateegia kiireks sudoku lahendamiseks ning graafiline kasutajaliides, mis abistab kasutajat sudoku lahendamisel.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 25 leheküljel, 3 peatükki, 23 joonist, 1 tabelit.

Abstract

Optimising Sudoku Solver for Speed

The aim of this thesis was to implement a Sudoku Solver with graphical user interface in Java programming language which is optimised for speed.

During writing of this thesis different solving techniques were investigated to solve Sudoku. In this thesis naked and hidden single from logical solving techniques are used. In addition to logical solving techniques backtracking is used as a solving technique.

Different solving strategies were tested using described solving techniques. The aim of testing was to find a fast Sudoku solving strategy.

In this thesis a program for solving Sudoku was designed and implemented using the fastest solving strategy from the results of testing. Graphical user interface is also implemented in the program from which user can insert new sudoku and make changes to it. User can solve the sudoku from the graphical user interface and as a result solving speed and used logical solving techniques as steps are displayed to the user. User can also ask for clues and check for conflicts with the digits on Sudoku from the graphical user interface.

As a result of this thesis an optimised Sudoku solving strategy was developed using algorithms described in this thesis with graphical user interface which helps the end user to solve Sudoku.

The thesis is in Estonian and contains 25 pages of text, 3 chapters, 23 figures, 1 tables.

Lühendite ja mõistete sõnastik

s	sekund
ms	millisekund
MVC	*Model View Controller*
vt	*vaata*

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Käesolevas töös käsitletakse üheksa korda üheksa sudoku lahendajat. Sudoku on loogikal põhinev puzzlemäng. Mäng koosneb üheksa korda üheksa ruudustikust, mis on jaotatud ridadeks, veergudeks ja kolm korda kolm regioonideks. Mängu eesmärgiks on paigutada igasse ruutu number üks kuni üheksa nii et igas reas, veerus ja regioonis esineb number ainult ühe korra.

Antud töö eesmärgiks oli realiseerida Java programmeerimiskeeles sudoku lahendaja koos kasutajaliidesega optimeerides sudoku lahendamise kiirusele. Kasutajaliideses on võimalik kasutajal sudoku peal opereerida erinevate protseduuridega nagu näiteks lahendamine ja vihjete andmine.

Töö esimeses osas kirjeldatakse erinevaid sudoku lahendamismeetodeid, mida töös kasutatakse sudoku lahendamiseks ning lahendusstrateegiate loomiseks. Töö teises osas kirjeldatakse sudoku lahendamiskiiruse optimeerimist ja testimise protsessi ning teostatakse erinevate lahendusstrateegiate tulemuste võrdlusi. Töö kolmandas osas kirjeldatatakse programmi üldist arhitektuuri ning realiseerimise käiku. Viimase käigus antakse ka ülevaade programmi toimimisest.

2 Sudoku lahendusmeetodid

Selles peatükis on toodud erinevad töös kasutatud sudoku lahendusmeetodid. Selleks et järgnevaid lahendusmeetodeid kasutada peab sudoku olema valiidne (vt. 2.1).

2.1 Valiidne sudoku

Sudoku on antud töö kontekstis valiidne, kui sellele leidub unikaalne lahendus ning on antud vähemalt seitseteist vihjet, mis on kõik valiidsed väärtused [1] .

Valiidne väärtus sudoku ruudul on selline, mis on number vahemikus üks kuni üheksa ning antud väärtus ei esine rohkem kui ühe korra väärtusega seotud reas, veerus ja regioonis.

Joonis 1. Näide mitte valiidsetest väärtustest sudokul (ruudud C4 ja C5).

2.2 Tagurdamise lahendusmeetod

Lahendusmeetodi eesmärgiks on leida rekursiivsel meetodil sudokule lahendus. Järgnevalt on kirjeldatud lahendusmeetodi tööpõhimõte. Lahendusmeetod leiab ilma väärtuseta ruudu ning proovib sellel numbreid vahemikus üks kuni üheksa kuni leiab valiidse väärtuse (vt. 2.1). Juhul kui lahendusmeetod leiab ruudule valiidse väärtuse, siis otsib lahendusmeetod järgmise ilma väärtuseta ruudu ning sooritab sama protseduuri. Juhul kui lahendusmeetod ei leia ruudule valiidset väärtust, siis liigub lahendusmeetod tagasi eelmise lahendusmeetodi poolt käideldud ruudu juurde ning leiab sellele järgmise valiidse väärtuse. Lahendusmeetod kordab kirjeldatud protsessi kuni kõik sudoku ruudud on täidetud [2] .

```
Find row, col of an unassigned cell
If there is none, return true

For digits from 1 to 9
   if there is no conflict for digit at row,col
         assign digit to row,col and recursively try fill in rest of grid
         if recursion successful, return true
         if !successful, remove digit and try another
if all digits have been tried and nothing worked, return false to trigger backtracking
```

Joonis 2. Tagurdamise lahendusmeetodi pseudokood [2] .

2.3 Loogilised lahendusmeetodid

Üksikkandidaadi lahendusmeetodi eesmärgiks on leida sudokul selline ilma väärtuseta ruut, millele on ainult üks võimalik kandidaat. Selleks uuritakse ruuduga seotud rea, veeru ja regiooni väärtuseid. Kui ruudule on ainult üks võimalik kandidaat, siis see määratakse ruudu väärtuseks.

Joonis 3. Näide ilma väärtuseta ruudust sudokul (E1), kus on ainult üks võimalik kandidaat (üksikkandidaat väärtusega 3).

Peidetud üksikkandidaadi lahendusmeetodi eesmärgiks on leida sudokul selline ilma väärtuseta ruut, millele on küll mitu kandidaati, aga uurides ruuduga seotud rea, veeru ja regiooni kandidaate saab välja selgitada õige kandidaadi. Kui mõni antud ruudu kandidaatidest esineb ainult ühe korra temaga seotud reas, veerus või regioonis, siis see määratakse ruudu väärtuseks.

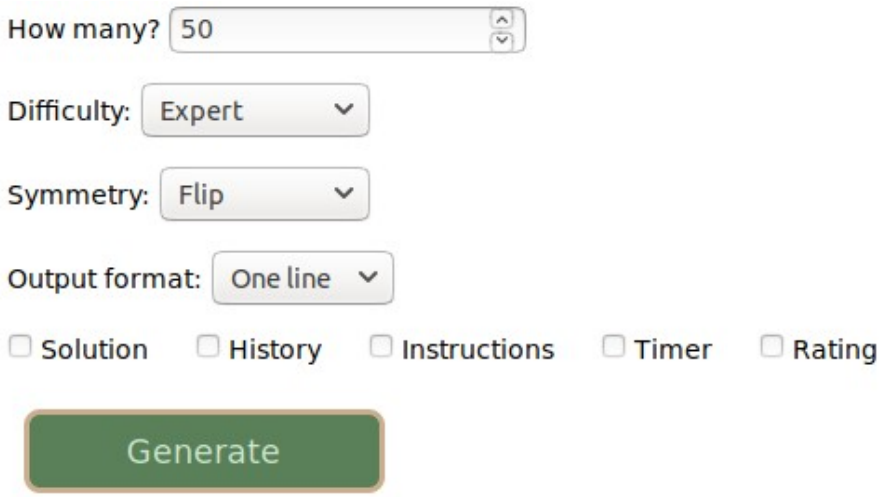
Joonis 4. Näide ilma väärtuseta ruudust sudokul (A5), kus on mitu kandidaati, aga ainult üks õige (peidetud üksikkandidaat väärtusega 8).

3 Testimine

Selles peatükis on kirjeldatud töös tehtud testimise üldist protsessi ning toodud erinevate katsete tulemused ja järjeldused.

3.1 Testimise üldine protsess

Selleks, et uurida erinevate lahendusstrateegiate mõju sudoku lahendamiskiirusele on vaja testülesandeid, mille peal lahendusstrateegiad rakendada. Selleks on töös kasutatud veebipõhist vaba tarkvara [3] , millega on koostatud kakssada viiskümmend erinevat sudoku testülesannet kõige suurema raskusastmega (*Difficulty: Expert)*. Kasutatud tarkvaras on võimalik koostada erineva sümmeetriaga sudoku ülesandeid. Töös on koostatud iga sümmeetria valiku kohta viiskümmend testülesannet.



Joonis 5. viiekümne sudoku *Expert* raskusastmega *Flip* sümmeetriaga ülesannete genereerimine qqwing tarkvaraga [3] .

Testimise töövoog on selline, et lahendatakse testülesandeid kümme korda ning pärast igat lahendamise tsüklit salvestatakse, kui kaua testülesannete lahendamine aega võttis.

Tulemustest võetakse keskmine, et minimeerida lahendamise protsessis tekkinud hälbeid. Testid jooksutatakse arvutil, millel ei tööta taustal muid programme.

3.2 Tagurdamise testimine

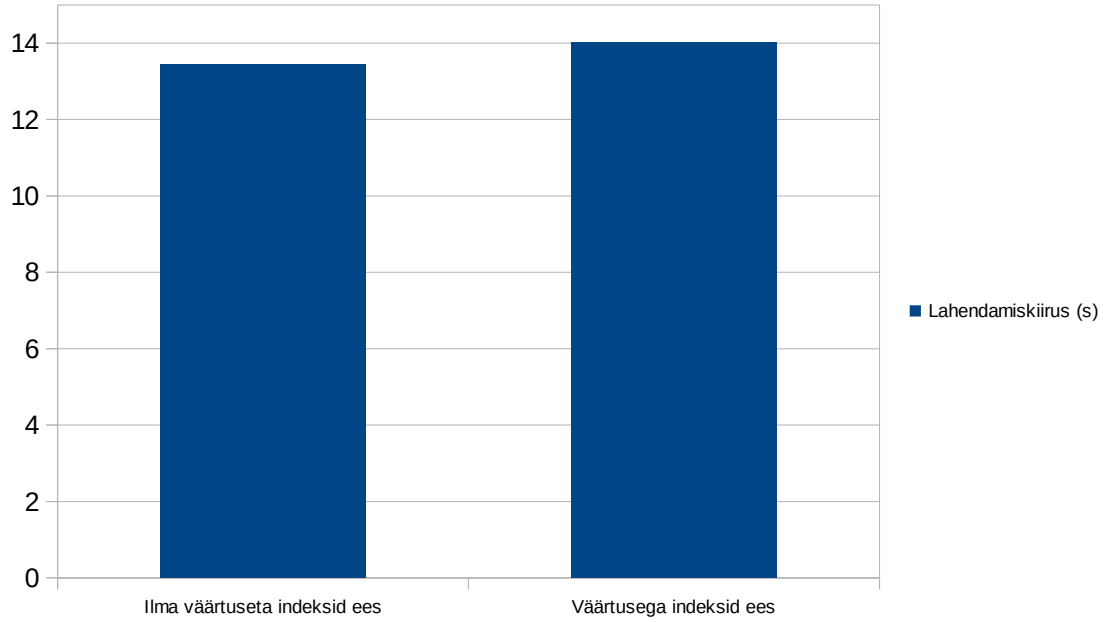
Kasutades tagurdamise (vt. 2.2) lahendusmeetodit vastavalt indeksite (vt. 4.2) vaikimisi järjestusele, kulus testülesannete lahendamiseks 13820ms (14s). Tegemist on kõige primitiivsema ja robustsema lahendusstrateegiaga, kus ei kasutata mitte mingisugust lahendamisele „intelligentset" lähenemist.

Kuna lahendusmeetod kasutab ühelt indeksilt teisele liikumiseks indeksite itereerimist, siis uuriti järgnevalt kas sellel, millises järjekorras lahendusmeetod indekseid itereerib on mõju lahendamiskiirusele. Enne lahendusmeetodi kasutamist järjestati indeksid vastavalt järgnevates peatükkides kirjeldatule.

3.2.1 Ilma väärtuseta ja väärtusega indeksite järjestus

Esiteks järjestati väärtusega indeksid kõige ette ning nende järele ilma väärtuseta indeksid. Ilma väärtuseta indeksite omavaheline järjestus sõltus sellest, millises järjekorras ilma väärtuseta indeksid esinesid vastavalt vaikimisi paigutusele. Selle tulemusena kulus lahendusmeetodil testülesannete lahendamiseks 14019ms (14s).

Teiseks järjestati indeksid vastupidi, kus ilma väärtuseta indeksid järjestati kõige ette ning nende järele väärtusega indeksid. Selle tulemusena kulus testülesannete lahendamiseks 13428ms (13s).



Joonis 6. Väärtusega ja ilma väärtuseta indeksite omavahelise järjestuse tulemused.

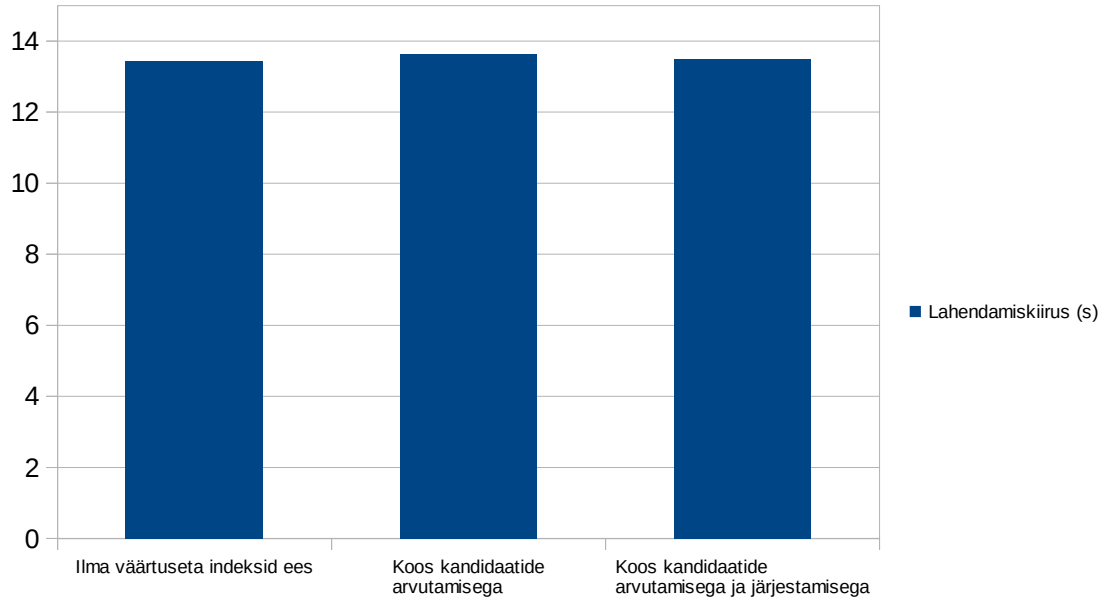
Väärtusega ning ilma väärtuseta indeksite omavaheline järjestuse mõju lahendamiskiirusele ei ole küll drastiline kuid siiski märkimisväärne. Pigem on oluline järjestada ilma väärtuseta indeksid kõige ette selleks, et vältida lahendamise aeglustamist kuna lahendusmeetod peab itereerima rohkem indekseid kui tegelikult tarvis. Tegelikult võib mõelda ka nii, et mingil määral eemaldati lahendusmeetodist otsimise protseduur, sest ilma väärtuseta indeksid on itereerimisel kõige ees ning seetõttu ei itereerita väärtusega indekseid, millega pole lahendusmeetodil midagi teha.

Järgnevatel töös tehtud testimistel on indeksid juba järjestatud nii, et ilma väärtuseta indeksid on kõige ees, sest vastavalt selles peatükis uuritud tulemustele on nii testülesannete lahendamine kiirem.

3.2.2 Ilma väärtuseta indeksite järjestus

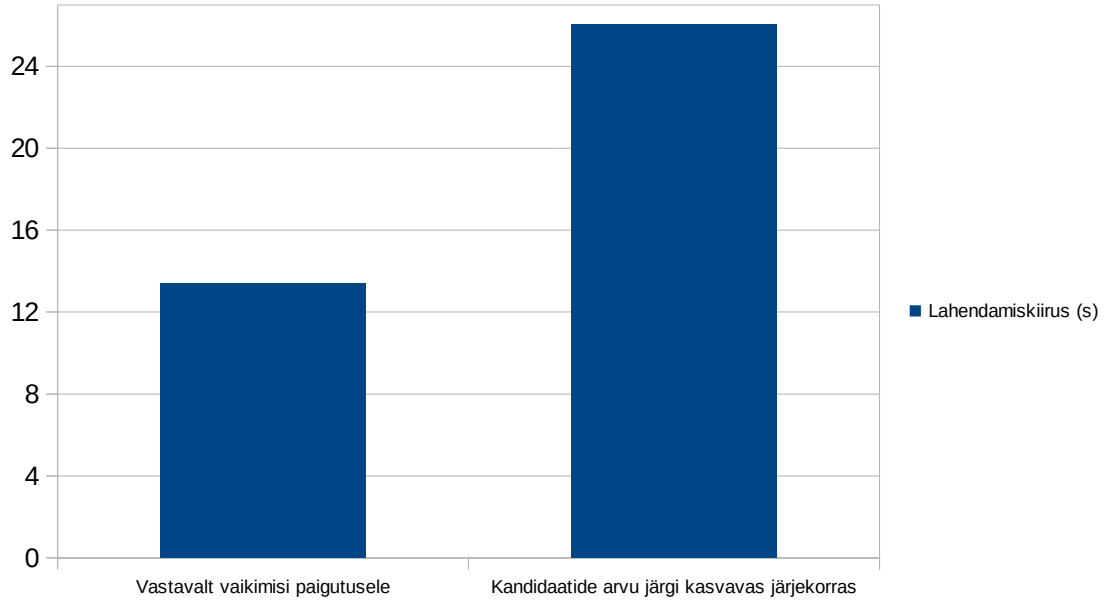
Selles peatükis uuriti ilma väärtuseta indeksite omavahelise järjestuse mõju lahendamiskiirusele. Eelkõige uuriti, kui kaua võtab aega kõikide kandidaatide arvutamise protseduur. Selleks arvutatakse enne lahendusmeetodi kasutamist kõikidele ilma väärtuseta indeksitele kandidaadid. Koos ilma väärtuseta indeksitele kandidaatide arvutamisega kulus testülesannete lahendamiseks 13643ms (14s). Võrreldes 3.2.1

tulemustega, kus parimal juhul kulus testülesannete lahendamiseks 13428ms (13s) võib sellest järeldada, et ilma väärtuseta indeksitele kandidaatide arvutamine ei ole väga resursimahukas protsess. Lisaks sellele testiti ilma väärtuseta indeksite kandidaatide arvu järgi järjestamise protsessi resursimahukust. Koos ilma väärtuseta indeksitele kandidaatide arvutamisega ning nende kandidaatide arvu põhjal järjestamisega kulus testülesannete lahendamiseks 13488ms (13s). Sellest võib järeldada, et ka ilma väärtuseta indeksite järjestamine kandidaatide arvu järgi ei ole väga resursimahukas protsess.



Joonis 7. Kandidaatide arvutamise resursimahukuse tulemused.

Järgnevalt järjestati ilma väärtuseta indeksid kandidaatide arvu järgi kasvavas järjekorras. See tähendab, et kui lahendusmeetod alustab itereerimist siis on need ilma väärtuseta indeksid kõige ees, millel on kõige vähem kandidaate. Selle tulemusena kulus testülesannete lahendamiseks 26045ms (26s). Sellest võib järeldada, et ilma väärtusteta indeksite järjestamine kandidaatide arvu järgi kasvavas järjekorras enne lahendusmeetodi kasutamist mõjutas lahendamiskiirust negatiivselt.



Joonis 8. Ilma väärtuseta indeksite järjestamine kandidaatide arvu järgi kasvavas järjekorras tulemused.

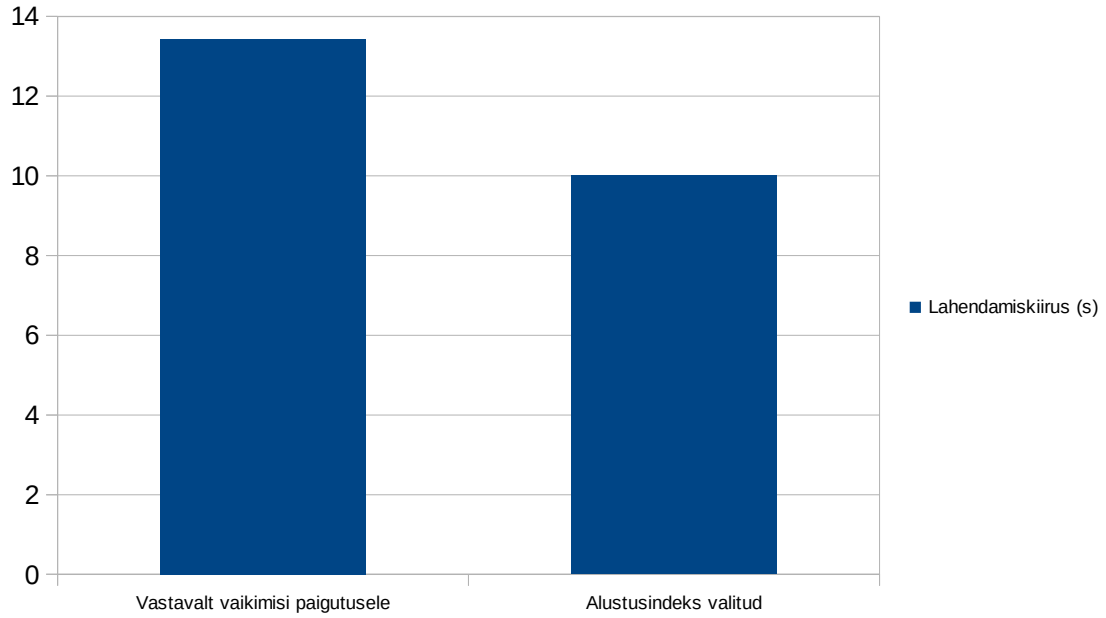
Võrreldes 3.2.1 parima tulemusega 13428ms (13s) kulus selles katses 12617 (13s) rohkem aega testülesannete lahendamiseks. Prooviti ka vastupidist lähenemist, kus järjestati ilma väärtuseta indeksid kandidaatide arvu järgi kahanevas järjekorras. Selle tulemusena kestis lahenduse leidmine ebamääraselt kaua ning autor otsustas testimise katkestada. Tehtud katsetest võib järeldada, et ilma väärtuseta indeksite itereerimise järjekorral kandidaatide arvu põhjal on mõju lahenduskiirusele.

3.2.3 Alustusindeks ning ilma väärtuseta indeksite järjestus

Selles peatükis uuritakse millisest ilma väärtuseta indeksist alustamine ning mis järjestuses järgnevate ilma väärtuseta indeksite itereerimine avaldab kõige suuremat positiivset mõju lahendamiskiirusele.

Esiteks katsetati alustamist sellelt ilma väärtuseta indeksilt, millel on kõige vähem kandidaate ning järgnevate ilma väärtuseta indeksite itereerimine toimub vastavalt sellele, mis järjekorras ilma väärtuseta indeksid esinesid vastavalt vaikimisi paigutusele. Juhul kui selliseid ilma väärtuseta indekseid on mitu kus kohast alustada, siis valitakse see ilma väärtuseta indeks alustusindeksiks, mis esineb kõige esimesena vastavalt

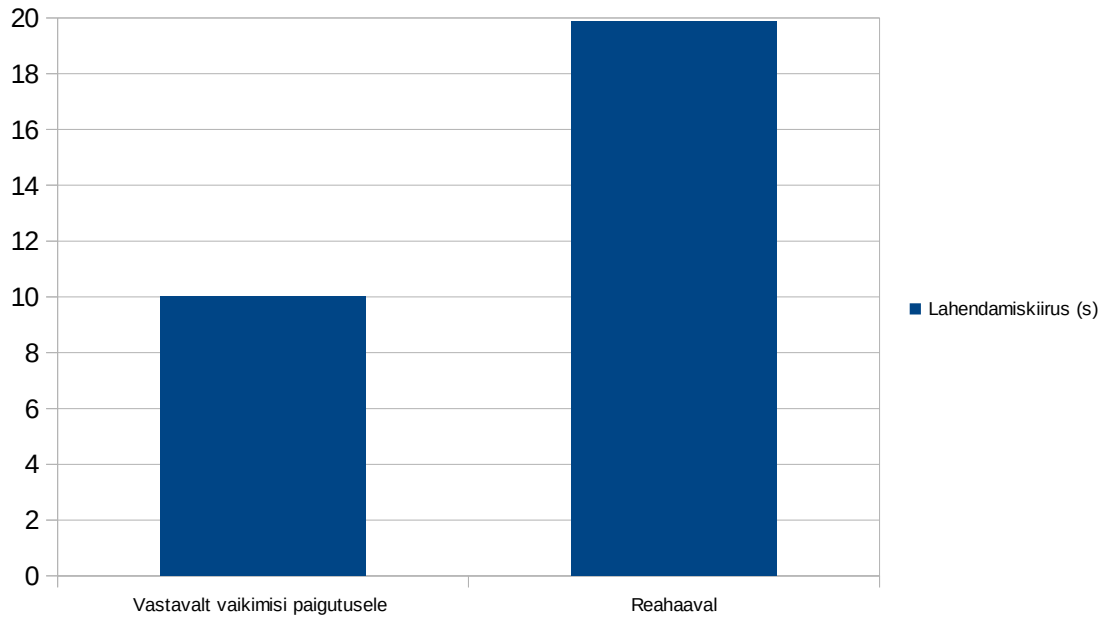
vaikimisi paigutusele. Selle tulemusena kulus testülesannete lahendamiseks 10017ms (10s).



Joonis 9. Alustusindeksiks on vähima kandidaatide arvuga ilma väärtuseta indeks tulemused.

Võrreldes parima tulemusega peatükis 3.2.1 milleks oli 13428ms (13s) on selles katses kirjeldatud ilma väärtuseta indeksite järjestusel märkimisväärne mõju lahendamise kiirusele ning testülesanded lahendati 3411ms (3s) kiiremini. Sellest võib järeldada, et kui alustada sellelt ilma väärtuseta indeksilt, millel on kõige vähem kandidaate ning järgnevate ilma väärtuseta indeksite itereerimine toimub vastavalt vaikimisi paigutusele, siis leitakse lahendus kiiremini.

Teisena katsetati alustamist sellelt ilma väärtuseta indeksilt, millel on kõige vähem kandidaate ning järgnevate ilma väärtuseta indeksite itereerimine toimub alustusindeksilt reahaaval edasi liikudes. Selle tulemusena kulus testülesannete lahendamiseks 19855ms (20s).



Joonis 10. Alustusindeksist järgnevate ilma väärtuseta indeksite itereerimise järjekorra tulemused.

Võrreldes eelmises lõigus toodud tulemustega kulus testülesannete lahendamiseks selles lõigus kirjeldatud strateegiaga 9838ms (10s) rohkem aega. Sellest võib järeldada, et sellel mis järjekorras pärast alustusindeksi valimist järgnevaid ilma väärtuseta indekseid itereeritakse on suur mõju lahenduskiirusele. Antud katse tulemusena ei ole hea variant järgnevaid ilma väärtuseta indekseid itereerida alustusindeksilt reahaaval edasi liikudes.

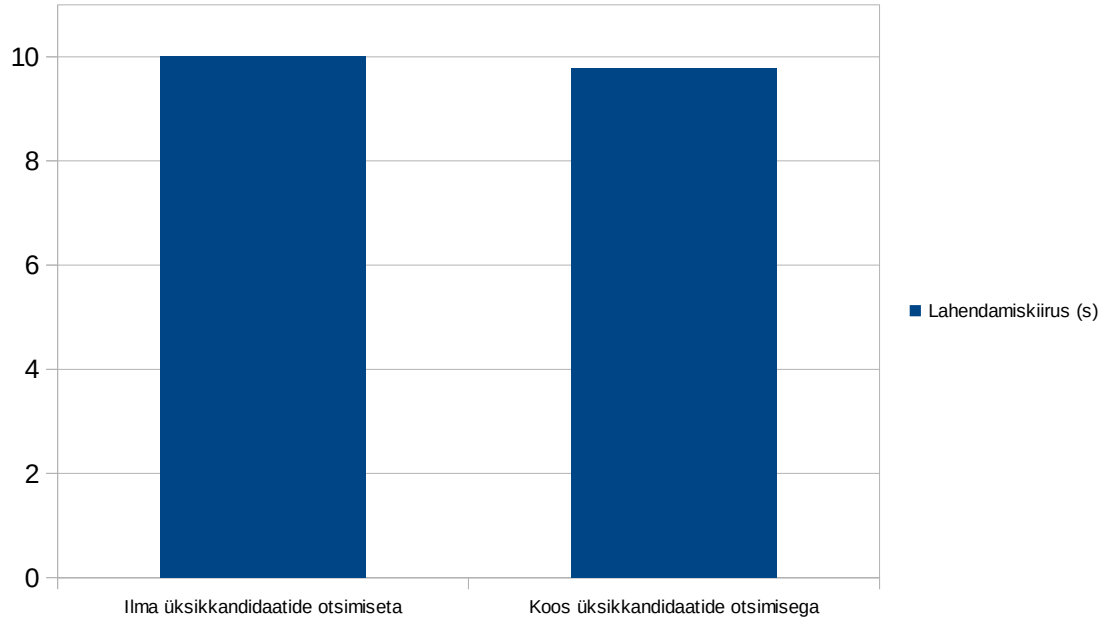
Antud peatükkides uuritud tulemustest võib järeldada, et antud töö kontekstis kõige kiirem variant testülesannete lahendamiseks kasutades tagurdamise lahendusmeetodit on valida alustusindeksiks selline ilma väärtuseta indeks, kus on kõige vähem kandidaate või kui neid on mitu siis valida see ilma väärtuseta indeks, mis on kõige esimene vastavalt vaikimisi paigutusele. Järgnevad ilma väärtuseta indeksid järjestatakse vastavalt vaikimisi paigutusele ning nende järele väärtusega indeksid. Järgnevates testimistes ka sellist järjestust kasutatakse.

3.3 Üksikkandidaadi lahendusmeetodi lisamine ja testimine

Selles peatükis rakendatatakse testülesannete peal lisaks tagurdamise (vt. 2.2) lahendusmeetodile ka üksikkandidaadi (vt. 2.3) lahendusmeetodit.

Eelkõige uuriti üksikkandidaadi lahendusmeetodi resursimahukust. Koos üksikkandidaadi lahendusmeetodiga, mis otsis välja kõik need ilma väärtuseta indeksid kuhu saab vastavalt 2.3 toodud tingimustele väärtuse sisestada enne tagurdamise lahendusmeetodi kasutamist kulus testülesannete lahendamiseks 9790ms (10s). Keskmiselt leidis üksikkandidaadi lahendusmeetod igas testülesandes neli üksikkandidaati.

Tuleb ka arvestada sellega, et ilma väärtuseta indeksitele üksikkandidaatide väärtustamine võib paljastada antud ilma väärtuseta indeksiga seotud ilma väärtuseta indeksite kandidaatide eemaldamise tagajärjel veel üksikkandidaate.



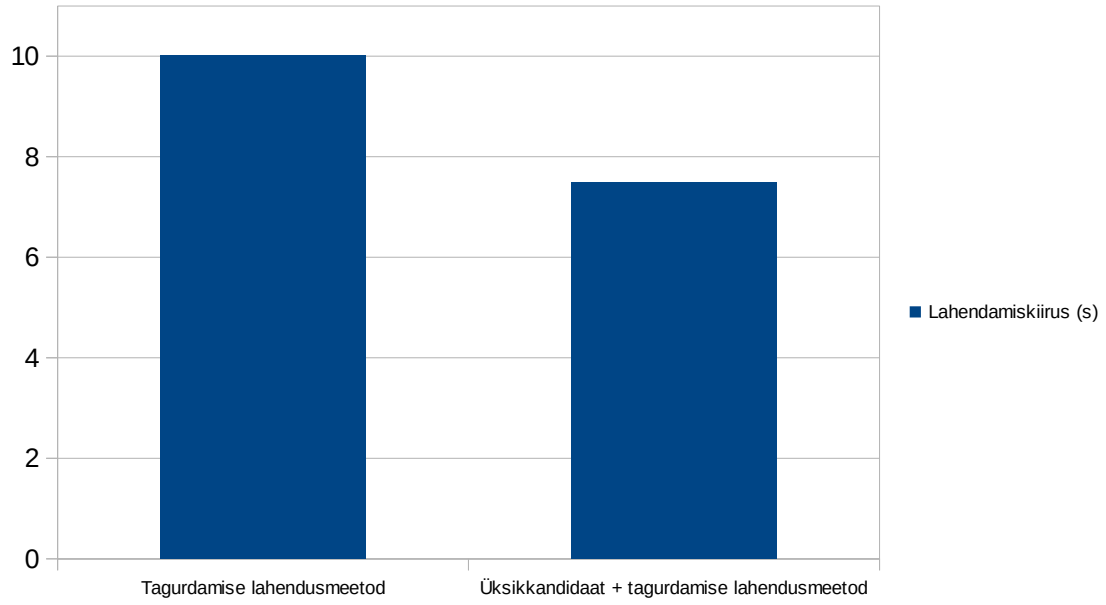
Joonis 11. Üksikkandidaadi otsimise resursimahukuse tulemused.

Võrreldes 3.2.3 parima tulemusega 10017ms (10s) oli testülesannete lahendamine 227ms (0,02s) kiirem. Sellest võib järeldada, et üksikkandidaadi lahendusmeetodi resursimahukus on väga väike ning seetõttu tuleks üksikkandidaadi lahendusmeetodiga leida nii paljudele ilma väärtuseta indeksitele väärtused kui võimalik, et lahenduse leidmine oleks võimalikult kiire.

3.3.1 Üksikkandidaadi lahendusmeetodi rakendamine

Järgmiseks uuriti üksikkandidaadi ja tagurdamise lahendusmeetodite rakendamist testülesannete peal. Lahendamise alguses kasutatakse üksikkandidaadi

lahendusmeetodit, et leida nii paljudele ilma väärtuseta indeksitele väärtused kuni see enam pole antud lahendusmeetodiga võimalik. Kui leidub veel ilma väärtuseta indekseid ehk lahendust pole veel leitud, siis jätkatakse lahendamist tagurdamise lahendusmeetodiga kuni leitakse lahendus. Selle tulemusena kulus testülesannete lahendamiseks 7488ms (7s).



Joonis 12. Üksikkandidaadi ja tagurdamise lahendusmeetodi rakendamise tulemused.

Võrreldes 3.2.3 parima tulemusega 10017ms (10s) on lisaks üksikkandidaadi lahendusmeetodiga testülesannete lahendamine 2529ms (2,5s) kiirem. Kuna üksikkandidaadi lahendusmeetodi resursimahukus on väga väike siis võib sellest järeldada, et lahenduse leidmine oli kiirem sellepärast, et tagurdamise lahendusmeetod pidi itereerima vähem ilma väärtuseta indekseid.

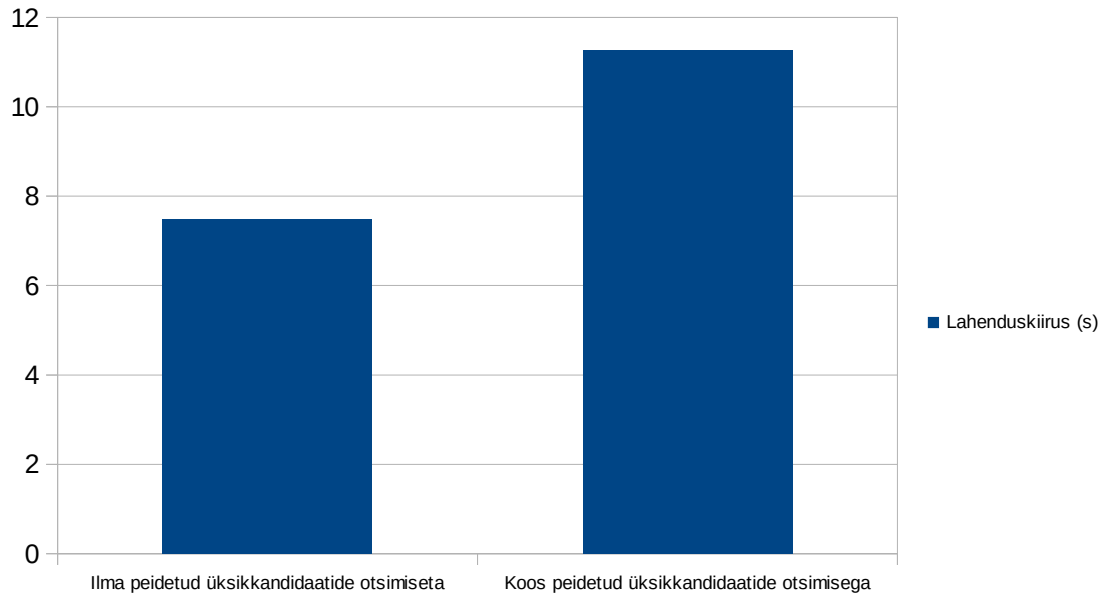
Üksikkandidaadi lahendusmeetodi resursimahukus on väga väike ning seetõttu peaks seda lahendusmeetodit kasutama võimalikult tihti selleks, et leida nii paljudele ilma väärtuseta indeksitele väärtused kui võimalik. Selleks, et tagurdamise lahendusmeetod lõpetaks võimalikult kiiresti tuleks eelnevalt leida üksikkandidaadi lahendusmeetodiga võimalikult palju väärtuseid ilma väärtuseta indeksitele, et tagurdamise lahendusmeetod peaks vähem ilma väärtuseta indekseid itereerima.

3.4 Peidetud üksikkandidaadi lahendusmeetodi lisamine ja testimine

Selles peatükis rakendatatakse testülesannete peal lisaks tagurdamise (vt. 2.2) ning üksikkandidaadi (vt. 2.3) lahendusmeetodile ka peidetud üksikkandidaadi (vt. 2.3) lahendusmeetodit.

Eelkõige uuriti peidetud üksikkandidaadi lahendusmeetodi resursimahukust. Ennem peidetud üksikkandidaadi lahendusmeetodi rakendamist rakendati testülesannete peal üksikkandidaadi lahendusmeetodit. Koos peidetud üksikkandidaadi lahendusmeetodiga, mis otsis välja kõik need ilma väärtuseta indeksid kuhu saab vastavalt 2.3 toodud tingimustele väärtuse sisestada enne tagurdamise lahendusmeetodi kasutamist kulus testülesannete lahendamiseks 11263ms (11s). Keskmiselt leidis peidetud üksikkandidaadi lahendusmeetod igas testülesandes viis peidetud üksikkandidaadi tingimustele vastavat väärtust.

Tuleb ka arvestada sellega, et ilma väärtuseta indeksitele peidetud üksikkandidaatide väärtustamine võib paljastada antud ilma väärtuseta indeksiga seotud ilma väärtuseta indeksite kandidaatide eemaldamise tagajärjel veel üksikkandidaate või peidetud üksikkandidaate.

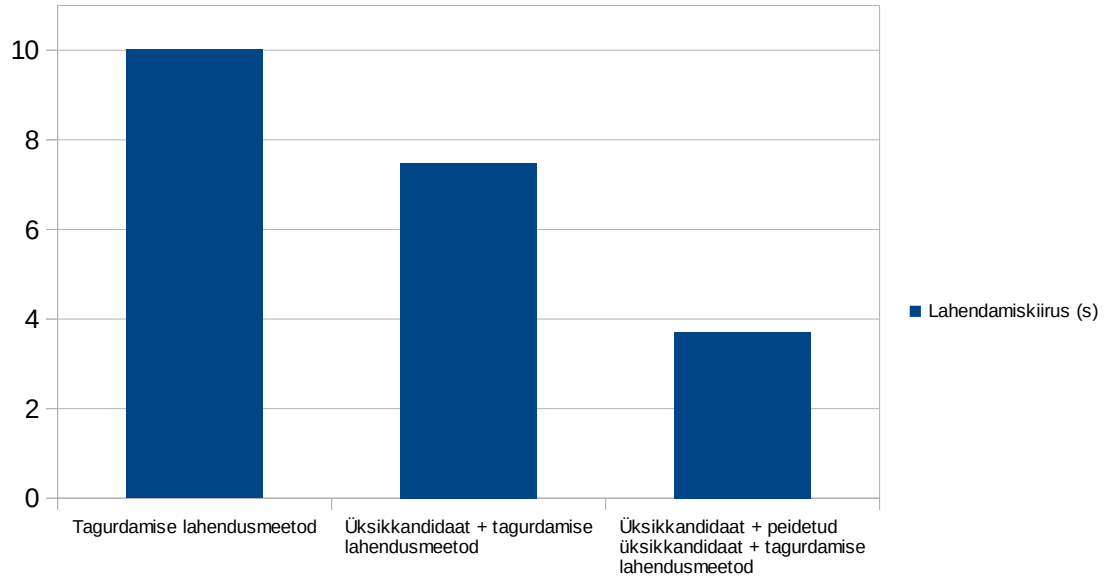


Joonis 13. Peidetud üksikkandidaadi resursimahukuse tulemused.

Võrreldes 3.3.1 parima tulemusega 7488ms (7s) oli testülesannete lahendamine 3775ms (4s) aeglasem. Sellest võib järeldada, et peidetud üksikkandidaadi lahendusmeetodi resursimahukus on väga suur.

3.4.1 Peidetud üksikkandidaadi lahendusmeetodi rakendamine

Järgmiseks uuriti üksikkandidaadi, peidetud üksikkandidaadi ja tagurdamise lahendusmeetodite rakendamist testülesannete peal. Lahendamise alguses kasutatakse üksikkandidaadi lahendusmeetodit, et leida nii paljudele ilma väärtuseta indeksitele väärtused kuni see enam pole antud lahendusmeetodiga võimalik. Kui leidub veel ilma väärtuseta indekseid ehk lahendust pole veel leitud, siis jätkatakse lahendamist peidetud üksikkandidaadi lahendusmeetodiga. Kui leidub veel ilma väärtuseta indekseid, siis leitakse lahendus tagurdamise lahendusmeetodiga. Selle tulemusena kulus testülesannete lahendamiseks 3717ms (4s).



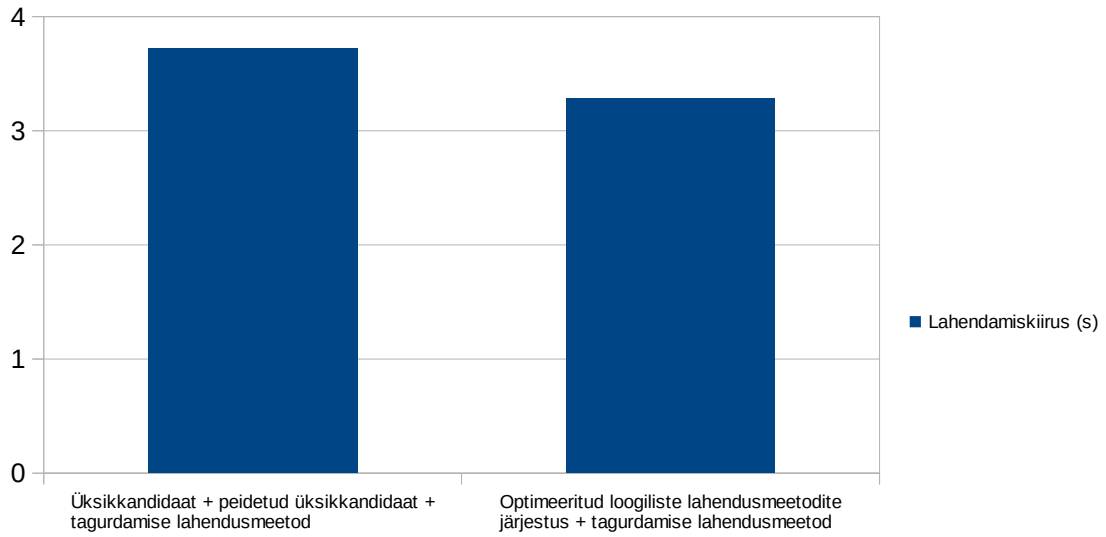
Joonis 14. Üksikkandidaadi, peidetud üksikkandidaadi ja tagurdamise lahendusmeetodi rakendamise tulemused.

Võrreldes 3.3.1 parima tulemusega 7488ms (7s) oli testülesannete lahendamine 3771ms (4s) kiirem. Kuigi peidetud üksikkandidaatide lahendusmeetodi resursimahukus on väga suur, siis on see ikkagi rakendamist väärt, sest tagurdamise lahendusmeetod lõpetab palju kiiremini. Tagurdamise lahendusmeetod lõpetab palju kiiremini just sellepärast, et

üksikkandidaatide ja peidetud üksikkandidaatide lahendusmeetodid leiavad juba omajagu ilma väärtuseta indeksitele väärtused ning seetõttu peab tagurdamine vähem ilma väärtuseta indekseid itereerima.

3.5 Loogiliste lahendusmeetodite järjestus

Kuna üksikkandidaadi lahendusmeetodi resursimahukus on peatükis 3.3 toodud tulemustele vastavalt väga väike ning peidetud üksikkandidaadi resursimahukus on peatükis 3.4 toodud tulemustele vastavalt väga suur siis prooviti teha lahendamist kiiremaks niimoodi, et muudeti ära üksikkandidaadi ja peidetud üksikkandidaadi lahendusmeetodite teostamise järjestus. Nimelt igakord, kui leitakse peidetud üksikkandidaadi lahendusmeetodi teostamise ajal veel üksikkandidaate, siis leitakse jälle kõik üksikkandidaadid üles ennem peidetud üksikkandidaadi lahendusmeetodi jätkamisega. Selle tulemusena kulus testülesannete lahendamiseks 3281ms (3s).



Joonis 15. Optimeeritud loogiliste lahendusmeetodite järjestuse rakendamise tulemused.

Võrreldes 3.4.1 parima tulemusega 3717ms (4s) oli eelnevas lõigus kirjeldatud lahendamisstrateegia 436ms (0,4s) kiirem. Sellest võib järeldada, et loogiliste lahendusmeetodite teostamise järjekorral on märkimisväärne mõju lahendamiskiirusele.

3.6 Testimiste kokkuvõte

Testimiste tulemusena osutus kõige kiiremaks lahendusstrateegiaks kasutada loogilisi lahendusmeetodeid kuni võimalik eelistades alati üksikkandidaatide lahendusmeetodit seejärel tagurdada. Tagurdamisel vastavalt vaikimisi paigutusele ilma väärtuseta indeks, millel on kõige vähem kandidaate järjestatud esimeseks ja selle järele ilma väärtuseta indeksid ning nende järel väärtusega indeksid.

Tabel 1. Testimiste tulemuste kokkuvõte.

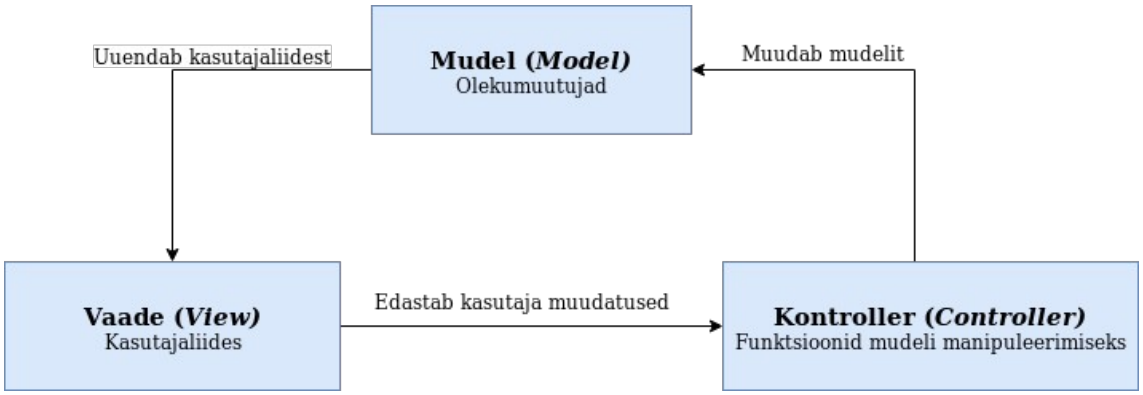
Lahendusstrateegia	**Lahendamiskiirus (s)**
Tagurdamisel väärtusega indeksid järjestatud ette ning nende järele ilma väärtuseta indeksid.	14,02
Tagurdamisel ilma väärtuseta indeksid järjestatud ette ning nende järele väärtusega indeksid.	13,43
Tagurdamisel ilma väärtuseta indeksid järjestatud ette ja ilma väärtuseta indeksid järjestatud omavahel kandidaatide arvu järgi kasvavas järjekorras ning nende järele väärtusega indeksid.	26,05
Tagurdamisel vastavalt vaikimisi paigutusele ilma väärtuseta indeks, millel on kõige vähem kandidaate järjestatud esimeseks ja selle järele ilma väärtuseta indeksid ning nende järel väärtusega indeksid.	10,02
Ennem tagurdamist otsitakse kõik üksikkandidaadid ning seejärel tagurdatakse. Tagurdamisel vastavalt vaikimisi paigutusele ilma väärtuseta indeks, millel on kõige vähem kandidaate järjestatud esimeseks ja selle järele ilma väärtuseta indeksid ning nende järel väärtusega indeksid.	7,49
Ennem tagurdamist otsitakse kõik üksikkandidaadid seejärel kõik peidetud üksikkandidaadid ning viimaseks tagurdatakse. Tagurdamisel vastavalt vaikimisi paigutusele ilma väärtuseta indeks, millel on kõige vähem kandidaate järjestatud esimeseks ja selle järele ilma väärtuseta indeksid ning nende järel väärtusega indeksid.	3,72

Lahendusstrateegia	**Lahendamiskiirus (s)**
Kasutatakse loogilisi lahendusmeetodeid kuni võimalik eelistades alati üksikkandidaatide lahendusmeetodit seejärel tagurdatakse. Tagurdamisel vastavalt vaikimisi paigutusele ilma väärtuseta indeks, millel on kõige vähem kandidaate järjestatud esimeseks ja selle järele ilma väärtuseta indeksid ning nende järel väärtusega indeksid.	3,28

4 Programmi arhitektuur ja realiseerimise käik

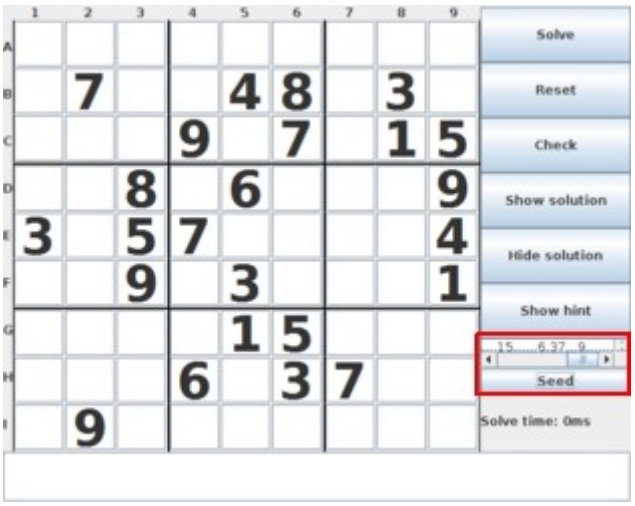
Selles peatükis on kirjeldatud programmi arhitektuuri ja realiseerimise käiku. Programmi üldine arhitektuur põhineb *Model View Controller (MVC)* [4] tarkvaraarenduse mustril.

MVC tarkvaraarendus mustri põhiliseks eesmärgiks on eraldada omavahel programmi andmed (*Model)*, liides andmete kuvamiseks/muutmiseks *(View)* ning andmete peal tehtavad protseduurid (*Controller)*. Sellise mustri kasutamise eeliseks on see, et komponendid on üksteisest sõltumatud ning seetõttu saab neid täiesti eraldi arendada.



Joonis 16. Programmis kasutatud *MVC* tarkvaraarendus muster.

Selleks, et alustada programmiga töötamist tuleb sisestada tekstivälja sudoku formaadis „..7.2.52...65.2..” kuni olemas on kaheksakümmend üks sümbolit, kus punkt tähistab ilma väärtuseta indeksit (vt. 4.2) ning number üks kuni üheksa väärtust sudoku ruudustikul. Sisestatud sümbolid kuvatakse järjest sudoku ruudustikule vastavalt vaikimisi paigutusele (vt. 4.2).

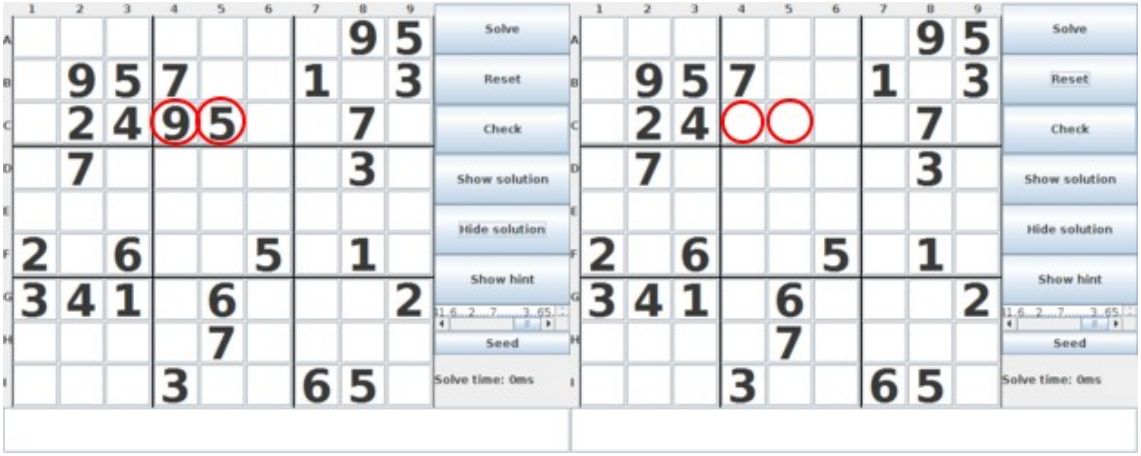


Joonis 17. Uue sudoku tekitamine kasutades tekstivälja ning vajutades „*Seed*" nuppu.

4.1 Olekumuutujad

Programm hoiab mälus erinevaid olekumuutujaid. Kõiki olekumuutujaid hoitakse mälus kuni programmi sulgemiseni. Järgnevalt on põhjendatud valitud olekumuutujate vajalikkus.

Esiteks, kui kasutaja tekitab uue sudoku sisestades selle õiges formaadis tekstivälja ning vajutades nuppu „*Seed"* (vt. 4), siis salvestatakse see esialgne sudoku sellisel kujul olekumuutujana mällu ning seda olekumuutujat enam programmi käigus ei muudeta välja arvatud, siis kui tekitatakse uus sudoku eelnevalt kirjeldatud protseduuri järgi. Seda on vaja sellepärast, et kui kasutaja tahab taastada algseisu ehk kõik muudatused tühistada vajutades „*Reset*" nuppu, siis oleks programmimälust võtta esialgselt tekitatud sudoku, mille pealt algseis taastada.



Joonis 18. „*Reset*” nupu vajutamisel kaotatud kasutaja muudatused indeksitelt C4 ja C5.

Olekumuutujana hoitakse programmimälus ka hetkel aktiivset kasutaja poolt tehtud muudatustega sudoku ruudustikku. Kõik protseduurid nagu näiteks väärtuste lisamine, lahendamine ja vihjete andmine toimub antud olekumuutuja peal.

Programmimälus hoitakse ka lahenduse leidmise sammude olekumuutujat. Lahendusesamm on realiseeritud programmis eraldi objektina, millel on kolm muutujat. Nendeks muutujateks on indeks (vt. 4.2), indeksil olev väärtus ning lahenduse tüüp. Kirjeldatud objektid paiknevad listingus ning kajastavad sudoku lahendamiseks kasutatud loogiliste lahendusmeetodite (vt. 2.3) samme. Juhul kui kasutaja vajutab nuppu „*Solve”*, siis kuvatakse kasutajale lahendamisel kasutatud loogilised lahendusmeetodite sammud (vt. Joonis 19).

Olekumuutujana on ka programmismälus sudoku lahendamiseks kulunud aeg. Juhul kui kasutaja vajutab nuppu „*Solve”*, siis lahendatakse sudoku ning kuvatakse antud olekumuutuja kasutajale (vt. Joonis 19).

Joonis 19. „*Solve*" nupu vajutamisel kuvatud lahendamisaeg (40ms) ning lahendamissammud kasutajale.

4.1.1 Vihjete andmine

Programmis on võimalik küsida vihjeid sudoku lahendamisel vajutades nupule „*Show hint*", mille tagajärjel tekib kasutajaliidesesse sudoku ruudustikule roheline indeks ning kasutajale kuvatakse sõnum vihje kohta. Vihjete andmine käib loogiliste lahendusmeetodite põhjal, see tähendab et kui sudokul pole loogilise lahendusmeetodiga lahendatavat indeksit, siis vihjet ei kuvata.

Joonis 20. Vihje kuvamine indeksile C5, kus on üksikkandidaat.

Selleks, et kuvada kõiki vihjeid kasutajale ükshaaval salvestatakse programmimällu olekumuutujana kõikide vihjete listing ning viimasena kuvatud vihje. Kõikide vihjete listing laetakse programmimällu ühe korra, siis kui kasutaja vajutab esimest korda vihjete andmise nuppu „*Show hint*”. Järgmisel nupu „*Show hint*” vajutamisel enam listingut uuesti ei tekitata vaid kuvatakse järgmine vihje eelmise kuvatud vihje asukoha järgi. Juhul kui kasutaja teeb sudokule muudatusi, siis tühjendatakse kõikide vihjete listing ning nuppu „*Show hint*” vajutamisel täidetakse vihjete listing uuesti. Eelnevat on vaja selleks, et vihjed oleksid aktiivse sudoku seisuga kooskõlas.

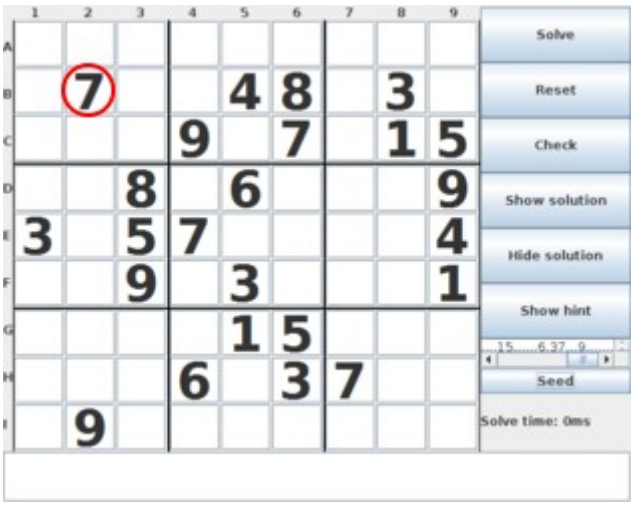
4.1.2 Konfliktide kuvamine

Kasutajal on võimalik kontrollida ka seda, kas hetkel aktiivsel sudokul on konflikte Nupule „*Check*” vajutades kuvatakse konfliktis olevad sudoku indeksid, kui selliseid on (vt. Joonis 21).

Joonis 21. „*Check*” nupule vajutades konfliktide kuvamine indeksitel A5, C6 ja C8.

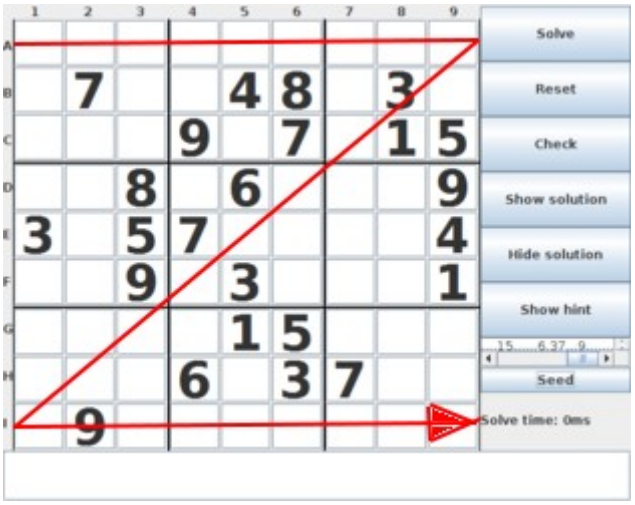
4.2 Indeks

Töös mõeldakse indeksi all mingit ruutu sudokul. Reatähed on tähestikulises kasvavas järjekorras ning veerunumbrid on numbrilises kasvavas järjekorras. Näiteks indeks „A1”, kus „A” tähistab rida ning „1” veergu on tegemist kõige ülemise vasakpoolsema ruuduga ning indeks „I9” viitab kõige alumisele parempoolsemale ruudule.



Joonis 22. Indeks "B2" väärtusega 7.

Indeksid paiknevad räsitabelis, kus vaikimisi paigutus tehakse sudoku kõige ülemisest vasakpoolsemast indeksist liikudes järjest reahaaval kuni kõige alumise parempoolsema indeksini.



Joonis 23. Indeksite vaikimisi paigutus (loetakse iga rida vasakult paremale üleval alla).

5 Kokkuvõte

Antud töö eesmärk sai täidetud ning Java programmeerimiskeeles realiseeriti sudoku lahendaja koos kasutajaliidesega, mis on optimiseeritud kiirusele. Selle jaoks teostati hulgaliselt testimisi ning valiti välja kõige kiirem sudoku lahendusstrateegia antud töö kontekstis.

Kasutajaliideses on funktsionaalsust, mis abistavad kasutajat sudoku lahendamisel nagu näiteks vihjete andmine ning konfliktide kontrollimine. Programmile valitud tarkvaradisaini muster võimaldab projekti lihtsasti edasi arendada ning lisada uut funktsionaalsust. Näiteks on võimalik asendada graafiline kasutajaliides täielikult *terminal*-il põhineva kasutajaliidesega ilma programmi loogikat muutmata.

Töö käigus õppis autor suuresti tarkvaradisaini mustrite rakendamist, et programmi arendamine oleks võimalikult sujuv ning arusaadav. Samuti õppis autor erinevate sudoku lahendusmeetodite rakendamist ning realiseerimist algoritmiliselt.

Kasutatud kirjandus

[1] Gary McGuire, Bastian Tugemann, Gilles Civario, „There is no 16-Clue Sudoku: Solving the Sudoku Minimum Number of Clues Problem via Hitting Set Enumeration," [Võrgumaterjal]. Available: https://arxiv.org/pdf/1201.0749.pdf. [Kasutatud 18 mai 2019].

[2] Julie Zelenski, „Exhaustive recursion and backtracking," [Võrgumaterjal]. Available: https://see.stanford.edu/materials/icspacs106b/H19-RecBacktrackExamples.pdf. [Kasutatud 18 mai 2019].

[3] Stephen Ostermiller, „QQWing Sudoku," [Võrgumaterjal]. Available: https://qqwing.com/generate.html. [Kasutatud 18 mai 2019].

[4] Tom Dalling, „Model View Controller Explained," [Võrgumaterjal]. Available: https://www.tomdalling.com/blog/software-design/model-view-controller-explained/. [Kasutatud 18 mai 2019].