TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Arvutitehnika instituut

Eero Rahamägi 123837

RUBIKU KUUBIKU LAHENDAMINE ERINEVAID ALGORITME KASUTADES

Bakalaureusetöö

Juhendaja: Sergei Kostin
PhD
Teadur

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: Eero Rahamägi

21.05.2017

Annotatsioon

Käesolevas bakalaureuse lõputöös kirjeldatakse erinevaid Rubiku kuubiku lahendamise algoritme. Analüüs ning võrdlus erinevate algoritmide vahel; kiirust ning optimaalsust arvesstades käikude arvu lühendamisel kuupi lahendades. Võrdlust aitab realiseerida ka prototüüpne programm mis lahendab kuubiku valitud algoritme kasutades.

Töö eesmärk on anda lugejale selge, arusaadav ning detailne ülevaade Rubiku kuubiku algoritmidest analüüsides ning kirjeldades erinevaid algoritme ning nende seotust keerukuse ning optimaalsuse lähtepunktist. Samuti annan ülevaate kasutatavast prototüüpsest programmist mis on võimeline lahendama Rubiku kuubikut suvalisest algolekust.

Lugejale esitatakse programne prototüüp visuaalsel kujul.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 31 leheküljel, 3 peatükki, 6 joonist, 1 tabel.

Abstract

Solving Rubiks Cube using different algorithms

This Bachelors thesis describes different Rubik's cube solving algorithms. Analysis and comparison between various algorithms; considering speed and optimality in decreasing the amount of steps it takes to complete a cube. A prototype program will be presented as a tool to help put the thesis into practice using selected algorithms.

However creating this kind of application can be quite complex taking into consideration the sheer size of algorithms and different patterns, searching methods and the complexity of the programming language itself. Developing this kind of program requires handling many different parts of code and algorithms simultaneously. There are a number of ways to achieve this feat but on a design point of view, the most important parts were picked out to merge within the application.

The design process involves creating a detailed description of the program and the graphical user interface in a way that is utmost understood by anyone that glances upon the interface.

This thesis is structured into three chapters, the first one being about the methods and approach of methods, which one to use and studying onwards by learning and adapting new methods. The second chapter describes the optimal solutions by algorithms and comparison between them. The last chapter focusing on the programming and desing part of the program itself.

The first chapter

The purpose of this thesis is to give the reader a clear, understandable and detailed overview of Rubik's cube algorithms by analysing and and describing different algorithms and their connectivity in difficulty and optimality. In addition I'll be giving an overview of the prototype which is capable of solving the Rubik's cube from any given state.

The reader will be presented a prototype in visual form.

The thesis is in estonian and contains 31 pages of text, 3 chapters, 6 figures, 1 table.

Sisukord

Jooniste loetelu

Tabelite loetelu

Sissejuhatus

Rubiku kuup on kolmedimensiooniline puslelaadne mänguasi, mis leiutati 1974 aastal hungarlasest skulptori ning arhitektuuriprofessori Ernõ Rubik poolt. Klassikaline Rubiku kuup koosnes kolmest kihist, iga kiht koosnes üheksast kuubikust mis olid omavahel ühendatud 3x3 võrgustikus. Sellest tulenevalt omas kuup kuut tahku, iga tahk üheksat kuubikut. Iga tahk oli eristamiseks värvitud eri värvidega: roheline, valge, punane, sinine, kollane ning oranž. Mänguasja eesmärk on ärasegatud kuup tuua sellisse asendisse, kus kõigil kuuel tahul on vaid ühesugused värvid, kasutades liigutusi mis keeravad tahkusid päri- ja vastupäeva.

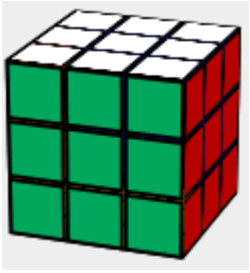
Rubiku kuubik pole tänapäevalgi oma populaarsust kaotanud. Võttes arvesse, et WCA (World Cube Association) korraldab turniire ning võistlusi selgitamaks parimatest parimaid varieeruvates kategooriast alustates kiiremaist lahendamisest lõpetades kinnisilmi lahendamisega. *Speedcubing*u nime alla kuuluvad mitmed variatsioonid originaalsest Rubiku kuubikust: *Pocket* Cube 2x2x2, Rubik's Cube 3x3x3, Rubik's Revenge 4x4x4, Professor's Cube 5x5x5 ning V-Cube 6, 7 ja 8 vastavalt 6x6x6, 7x7x7 ja 8x8x8 kuubid.

Mitmed veebilehed internetis pakuvad automaatset lahendamist võttes sisendiks ära segatud Rubiku kuubi ning lahendamisalgoritme kasutades lahendab samm-sammu haaval enne antud sisendiga kuubi.

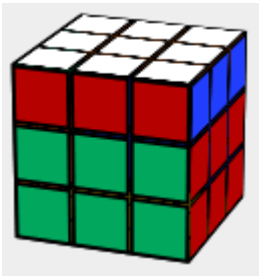
Minu lõputöö eesmärgiks oli luua sarnane programm/prototüüp mis lahendab kuubi mitte küll ühte ja sama algorimi kasutades, vaid valides ise oma algoritmi. Lisaks sellele on võimalik kasutada manuaalset sisendit kus kasutaja saab ise nupuvajutuste abil liigutada tahkusid oma tahte järgi.

Lühendite ja mõistete sõnastik

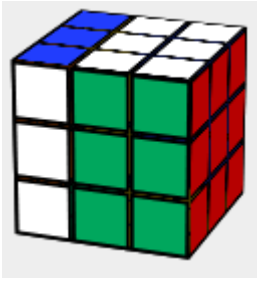
Algolek



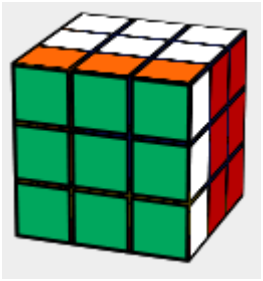
U (up) Pealmise tahu keeramine päripäeva 90°



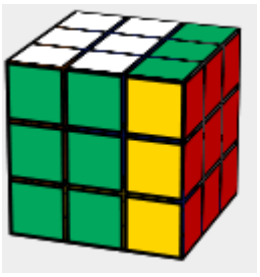
L (left) Vasaku tahu keeramine päripäeva 90°



F (front) Eesmise tahu keeramine päripäeva 90°

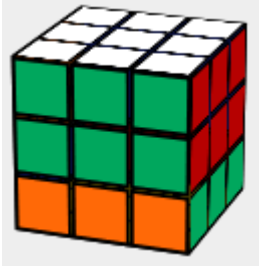


R (right) Parema tahu keeramine päripäeva 90°



B (back) Tagumise tahu keeramine päripäeva 90°

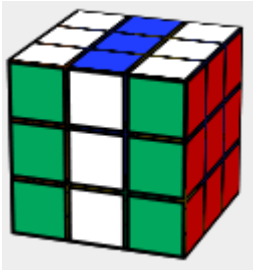
D (down) Alumine tahu keeramine päripäeva 90°



Järelliide ’ Tahu keeramine vastupäeva 90°

Järelliide 2 Tahu keeramine 180° (Näide:R2’ – Parema tahu keeramine vastupäeva 180°)

M (middle) Keskmise kihi (paralleelne R ja L tahuga) keeramine päripäeva 90°



E (equator) Keskmise kihi (paralleelne U ja D tahuga) keeramine päripäeva 90°

S (side) Keskmise kihi (paralleelne F ja B tahuga) keeramine päripäeva 90°

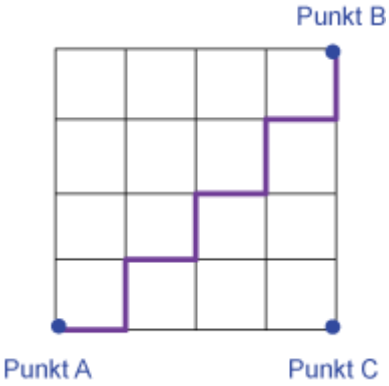
X Kuubi pööramine üle kujuteldava x-telje (R ja L jääb paigale) 90°

Y Kuubi pööramine üle kujuteldava y-telje (U ja D jääb paigale) 90°

Z Kuubi pööramine üle kujuteldava z-telje (F ja B jääb paigale) 90°

GUI Graphical User Interface, graafiline kasutajaliides

Manhattani kaugus [1] Kaugus kahe punkti vahel võrgustikus, mille liikumine toimub rangelt vertikaalset või horisontaalset teed pidi.



1. Meetodid lahendamiseks

1.1. Algaja meetod

Kasutab miinimum arv lihtsamaid järjestatud käike. Lahenduskäik on jagatud etappidesse kus iga etapp omab kindlat algoritmi tahkude lahendamiseks. Lahendus koosneb kolme kihi lahendamisest: alumine, keskmine ja pealmine. Esmalt valitakse üks värv ning võetakse aluseks pluss (+, cross) kujundi saavutamine. Seejärel lahendatakse selle tahu nurgad, positsioneerides need antud tahule ja seejärel orienteerides õigetesse nurkadesse. Järgmiseks pööratakse kuup pahupidi ning lahendatakse keskmist kihti kus põhirõhk saab olema vasaku ja parema pöörde peal.


Vasak: U’ L’ U L U F U’ F’


Parem: U R U’ R’ U’ F’ U F

Ilmuda võib ka variant kus servakuubik on tagurpidi:


Lahendus: kasutada kaks korda vasak/parempöörde algoritmi.

Kui kõik servad on lahendatud on tulemus järgmine:

Viimase kihi lahendamiseks võetakse jälle eesmärgiks luua pluss (+, cross). Viimasel kihil on plussi saamiseks kolm varianti: „L“ kujund, „miinus“ ja „pluss“. Saades „L“ kujundi võetakse appi algoritm „F R U R’ U’ F’“, et minna üle järgmisele kujundile, milleks on miinus. „L“ kujundilt „miinus“ kujundile üleminekuks kasutatakse algoritmi kaks korda. Saades „pluss“ kujundi tuleb veenduda tahku ümbritsevate kuubikute positsioonides. Kui kaks külgnevat tahku omavad vastupidiseid värve tuleb võtta kasutusele algoritm.


R U R’ U R U2 R’ U

Võib esineda juhtumeid kus algoritmi on vaja rohkem kui üks kord tarbida. Lõpptulemuseks on „pluss“ kujund mis on kujutatud pildil:

Viimaseks etapiks tuleb viimased nurgad positsioneerida ja seejärel orienteerida. Võttes suvalise positsioneeritud nurga ning sellele algoritmi „U R U’ L’ U R’ U’ L“ rakendades positsioneerime me ülejäänud kuubi nurgad õigetele positsioonidele. Kui ei leidu ühtegi nurka mis oleks positsioonis siis tuleb sama algoritmi ühe korra suvalise nurga peal rakendada ning see ilmutab vähemalt ühe positsioneeritud nurga millest saab lahendamist jätkata. Kui sama algoritmi rakendades saadi ülejäänud nurgad positsioonidele tuleb orienteerimiseks rakendada „R’ D’ R D“ algoritmi mis lahendab

ülejäänud nurgad. Meeles tuleb pidada, et iga kord kui nurk on lahendatud tuleb algoritmi lõppedes teha kas U või U’ pööre võttes ette uue lahendamata nurga. Algoritmi tuleb korrata kuni kuup laheneb.

1.2. Optimiseeritud algaja meetod

Optimiseeritud kiiruse ja käikude arvu järgi tuginedes meeldejätmisele ning laiendades järk-järgult edasijõudnud meetoditele.

Meetod tugineb tugevalt algaja meetodile laenates mitmeid algoritme just algaja meetodist. Lahendus hõlmab endas „valge pluss“-i lahendamist. Laenates „pluss“-i strateegiat algaja meetodist astume edasi situatsioonide juurde mis ilmuvad käikude optimiseerimisel.

F2 – sinivalge kuubik on vaja tuua pealmiselt tahult alumisele.

U’ R’ F R – sinivalge kuubik on vaja ümber pöörata, kuid peab jääma oma positsioonile.

F R2 D2 – kahe kuubiku liigutamine samaaegselt.

“Superflip“: R F L B R D –kõik 20 kuubikut on õiges permutatsioonis ja kaheks nurka on õigesti orienteeritud, aga kaksteist nurka valesti orienteeritud. [2]

1.3. Ortega meetod

Optimiseeritud mõistliku järjestuste arvu järgi, kiire tuvastamine ning pöörete tõhusus. Käikude piiratuse tõttu on kasutusel vaid U, F ja R pöörded ning kesksed liigutused. Põhirõhk on R küljel ainuüksi selle tõttu, et enamus inimesi on harjunud just seda külge kõige rohkem kasutama. Siiski on kõik järjestused minimaalsed, või ligilähedased sellele.

Antud lahenduskäik keskendub eelkõige positsioneerimisele kus idee peitub selles, et kuubikute permuteerimine on lihtsam kui nad on eelnevalt õigesti positsioneeritud. Kuubikuid orienteerides, kas siis enne või pärast positsioneerimist on alati lihtne, sest orienteerimine nõuab fokuseerimist ainult ühele värvile ja lahendusmustrile mida antud värv vajab. Keskmiste küljekuubikute permuteerimine viimasel kihil on samuti lihtne, sest printsiip mida eelnevalt kasutati kehtib ka siin. Keskmistele ja neid külgnevatele kuubikutele ei tasu tähelepanu pöörata. Servade lahendamiseks tuleb positsioneerida keskmised kuubikud, kuid piisab kui seda teha vaid pealmise ja alumise kihi jaoks. Kui teha seda iga külje jaoks, muudab see lahendamise veidi lihtsamaks, kuid sammude arv lahendamiseks kasvab. Kuubiku keskmised tükid positsioneeritakse koos neid külgnevate kuubikutega lahenduse viimases etapis.

Lahendamine Ortega meetodi järgi algab pealmise kihi nurkadest. Esmalt tuleb orienteerida nurgad vastavalt, et iga nurgakuubiku külgnevad kuubikd vastavad ülejäänud külgnevatele kuubikutele. Pärast ülemiste nurkade orienteerimist tuleb sama teha ka alumiste nurkadega pöörates kuubi pahupidi. Seejärel ilmub üks seitsmest asendist kus iga asendi jaoks on vastav algoritm nurkade paikaseadmiseks. Kui selle kihi nurgad on positsioneeritud tuleb need nüüd orienteerida vastavalt sellele mitu lahendatud paari tekkis. Jällegi on iga viie võimaliku väljundi jaoks oma algoritm 0-st paarist 4-ja paarini. Alates sellest hetkest kui nii pealmine kui alumine kiht on lahendatud tuleb positsioneerida iga tahu keskmine kuubik. Järgmine etapp haarab endas servade lahendamist kus vastavalt enda valitud meetodile lahendatakse servad. Lahendusse kuulub kolme pealmise serva, seejärel kolme alumise serva lahendamine, seejärel „veel-üks“ pealmine **või** alumine serv. Olles lahendanud oma valitud kas pealmise või alumise serva tuleb vastavalt viimane vastupidine serv lahendada kuubi täielikuks lahendamiseks. [3]

1.4. Waterman’i meetod

Üks keerulisemaid optimeeritud meetodeid, palju järjestusi mida meelde jätta.

Meetod baseerub „nurgad esimesena“ meetodil. Lahendamisel on kolm etappi:

Esimene etapp on ühe külje (kihi) lahendamine. Originaalis lahendatakse see „nurgad esimesena“ meetodi, seejärel kuubiku keskmine ja teda külgnevad tükid (+ kujutis).

Teises etapis lahendatakse nurgad viimasena ühe algoritmiga vastavalt kuubi asendile (mida on kokku 162 erinevat). Tulenevalt asendist on sellele vastav algoritm mis viib lahendatud kuubini. [4]

Viimases etapis lahendatakse kaks serva paremal (R) tahul, kuid samaaegselt lisades kolmanda serva kuhugi paremale tahule. Paremat tahku lõpetades tuleb orienteerida keskmise (M) servad, kõik ühes algoritmis. Antud samm on üsna keeruline (80 erinevat algoritmi), mis enamasti kasutab M, R, X ning U käike. Kuigi algoritmid on keerulised, käigud ise on kiired ja lihtsad. Viimase etapi viimane algoritm kujutab endas keskmise (M) kihi servade lahendamist, tegu on triviaalse variandiga, käike on vähe ning sellest tulenevalt on lahendamine kiire. [5]

2. Optimaalsed lahendused

2.1. Alampiir

Loendusalgoritmidega on tõestatud, et eksisteerivad positsioonid mis vajavad vähemalt 18 käiku, et lahendada. Seda tõestamaks tuleb lugeda kõik kuubi positsioonid mis eksisteerivad, seejärel loetleda positsioonide arv mis on võimalik saavutada maksimaalselt 17 käiguga.

2.2. Ülempiir

2.2.1. Jumala number (God's number)

Igat suvaolekus kuubikut on võimalik lahendada 20 või vähema käiguga. 1980 sätestati Jumala numbri alampiiriks 18 käiku. See saavutati tõhusalt analüüsides erinevaid käike mis hõlmasid 17 või vähem liigutust. Analüüsi tulemusel selgus, et selliseid järjestusi on vähem kui kuubiku enda positsioone. Esimeseks ülempiiriks oli 80 kui ilmusid esimesed voldikud kuubiku lahendamiseks. [6]

Date	Lower bound	Upper bound	Gap	Notes and Links
July, 1981	18	52	34	Morwen Thistlethwaite proves 52 moves suffice.
December, 1990	18	42	24	Hans Kloosterman improves this to 42 moves.
May, 1992	18	39	21	Michael Reid shows 39 moves is always sufficient.
May, 1992	18	37	19	Dik Winter lowers this to 37 moves just one day later!
January, 1995	18	29	11	Michael Reid cuts the upper bound to 29 moves by analyzing Kociemba's two-phase algorithm.
January, 1995	20	29	9	Michael Reid proves that the "superflip" position (corners correct, edges placed but flipped) requires 20 moves.
December, 2005	20	28	8	Silviu Radu shows that 28 moves is always enough.
April, 2006	20	27	7	Silviu Radu improves his bound to 27 moves.
May, 2007	20	26	6	Dan Kunkle and Gene Cooperman prove 26 moves suffice.
March, 2008	20	25	5	Tomas Rokicki cuts the upper bound to 25 moves.
April, 2008	20	23	3	Tomas Rokicki and John Welborn reduce it to only 23 moves.
August, 2008	20	22	2	Tomas Rokicki and John Welborn continue down to 22 moves.
July, 2010	20	20	0	Tomas Rokicki, Herbert Kociemba, Morley Davidson, and John Dethridge prove that God's Number for the Cube is exactly 20.

Tabel 1 ülem- ja alampiiride kujunemine üle aastate

2.2.2. Thistlethwaite-i algoritm

Thistlethwaite-i meetod erineb kihi- ja „nurgad esimesena“ algoritmidest selle poolest, et ta ei paiguta nurki õigetesse positsioonidesse ükshaaval. Selle asemel teeb ta seda kõikide nurgadega samaaegselt, piirates nurki aina vähematele võimalustele kuni leidub üks positsioon iga nurga jaoks ja kuup ongi lahendatud.

Selle saavutamiseks tehakse mõned käigud kuni ilmub positsioon mida saab lahendada veerapöördeid kasutamata U ja D tahkudel (kuigi poolpöördeid on vaja kasutada). Seejärel valitakse positsioon kus lahenduskäik ei kasuta U,D ega ka F ning B veerandpöördeid. Nende kitsendustega ilmub positsioon mis ei vaja neid eelpoolnimetatud veerandpöördeid, vaid lahendub hoopis poolpöörete abil.

Pesastatud gruppide abil on välja toodud grupid liigutuste jaoks ning nende positsioonide ilmnemise arv.

Esimene grupp G0=<L, R, F, B, U, D>, positsioonide arv $4{,}33*10^{19}$, faktor 2 048

Teine grupp G1=<L, R, F, B, U2, D2>, positsioonide arv $2{,}11*10^{16}$, faktor 1 082 565

Kolmas grupp G2=<L, R, F2, B2, U2, D2>, positsioonide arv $1{,}95*10^{10}$, faktor 29 400

Neljas grupp G3=<L2, R2, F2, B2, U2, D2>, positsioonide arv $6{,}63*10^{5}$, faktor 663 552

Viimane grupp on lahendud kuup ise, seega seda ei arvestata grupi liikmeks.

Liikumine lahendamise suunas toimub G0>G1>G2>G3>G4.

Esimese taseme faktor on 2 048 (2^{11}), see korrespondeerub sellele, et selle taseme eesmärk on parandada nurgade orientatsioon; on võimatu keerata nurki ainult poolpööretega U ja D tahkudel, kuid muidu saab kuubikuid panna igasse normaalsesse positsioon selle piiranguga.

Teise taseme faktor on 3^7 * 12!/(8!4!), see korrespondeerub sellele, et antud tase keskendub nurkade orientatsiooni parandamisele ja paigutab keskmist kuubikut külgnevad kuubikud õigesse positsiooni. Seda on lihtne ette kujutada kui on lubatud vaid poolpöörded F, B, U ning D tahkudel ning pool- ja veerandpöörded L ning R tahkudel. Sellest tulenevalt ei saa L ega R tahkudel korraga olla rohkem kui kaks värvi, lisaks sellele ei saa külgnevad kuubikud kunagi oma positsioonist väljuda ega vastasasendisse minna. Need piirangud loovad eelduse lahendamaks iga positsiooni antud tingimustel.

Kolmanda taseme faktor on $[8!/(4!4!)]^2*2*3$, see korrespondeerub sellele, et külgnevad kuubikud L ning R tahkudel on oma õigetes positsioonides, nurgad paigutatakse oma õigetesse nelikutesse, nurkade permutatsiooni paarsus on paaris ning terve keere iga neliku kohta on fikseeritud.

Viimane tase lahendab kuubi poolpööretega. [7]

2.2.3. IDA*

Iteratiivne süvendus A* (*Iterative deepening A**) on graafi liiklemine ja tee otsing mis suudab leida lühima tee määratud algussõlmest iga suvalise lõpplahendsõlmeni kaalutud graafis. Kuna tegu on sügavus-enne otsingualgoritmiga siis IDA* mälukasutus on märkimisväärselt väiksem A*. Erinevalt tavalisest iteratiivsest süvendatud otsingust, IDA* keskendub kõige lootusrikkamatele sõlmedele ning sellest tulenevalt ei liigu samale sügavusele igal pool otsingupuus.

IDA* on mälukitsendusega versioon A* st. IDA* suudab teha kõike mida A* suudab, sest sellel on A* optimaalne karakteristik leidmaks lühima tee oma otsipuus kasutamaks vähem mälu kui A*.

IDA* plussid:

- Leiab alati optimaalseima lahenduse eeldusel, et see eksisteerib ja kui heuristika on tagatud, peab see ka olema vastuvõetav.
- Heuristika pole vajalik, seda kasutatakse kõigest otsipuu protsessi kiirendamiseks.
- Eri heuristikaid võüib integreerida algorimi ilma, et oleks vaja muuta programmi baaskoodi.
- Iga käigu teekonnaväärtust võib kohandada algoritmi jaoks sama lihtsalt kui heuristikat.
- Kasutab märkimisväärselt vähem mälu mis kasvab lineaarselt tulenevalt sellest, et läbitud teekondi ei salvestata kui on saavutatud kindel teekonnasügavus.

IDA* miinused.

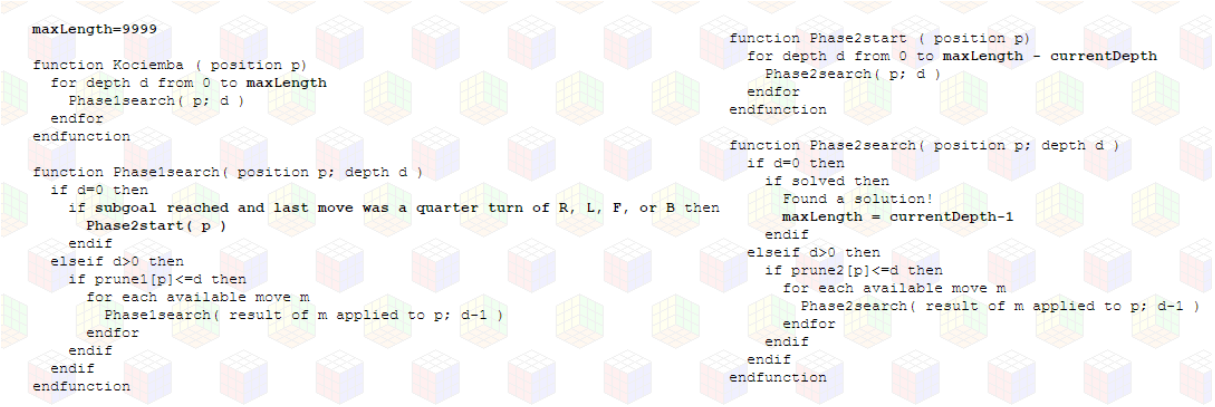
- Ei pea arvet juba külastatud teede üle ning sellest tulenevalt võib samu teid uuesti külastada.
- Aeglasem kuna kordab ning taasavastab juba avastatud teid.
- Nõuab rohkem protsessimisjõudlust kui A* [8]

2.2.4. Kociemba algoritm

Herbert Kociemba suutis kombineerida hulga ideid vägaefektiivsesse uude algoritmi mis annab kiiresti häid peaaegu optimaalseid lahendusi. Algoritm võib samahästi leida optimaalse tulemuse väga ruttu, kuid see võib võtta kaua aega tõestamaks, et lahendus on tõepoolest optimaalne teisi lühemaid lahendusi proovimata.

Esimene idee baseerus Thistlethwaite-i tööl. Kociemba kasutas küll kahte faasi nelja asemel. Tema esimene faas liigub G0-lt G1-le =<U, D, R2, L2, F2, B2> ja teine faas G1-lt G2-le = lahendatud kuup. Seega tulemuslikult kombineerib Kociemba Thistlethwaite-i faasid kokku. Sellegipoolest on ilmselge luua kärpetabeleid ning kasutada IDA* lahendamaks igat faasi. Ainuüksi see, et faaside arv on vähenenud kaks korda on kindel indikatsioon sellele, et täiendus Thistlethwaite-i algorimile eksisteerib, sest Kociemba esimene faas käib läbi Thistlethwaite-i esimesed kaks faasi, et leida lühim tee nende kahe faasi lahendamiseks. Sama käib ka Kociemba teise faasi kohta. Need faasid on arvutuslikult läbi kalkuleeritud ning nende maksimaalne pikkus leiti esimese ja teise faasi jaoks vastavalt 12 ja 18 käiku, seega kogu algoritm lahendub maksimaalselt 30 käiguga kasutades poolpöördeid. Lisaks on tõestatud, et käigud 29 ja 30 saab vahele jätta, tuues kogu käikude arvu 28-le. Antud algoritm oli pikka aega parim ülempiiri hoidja kuni leiti täiedavate meetoditega paremad algoritmid, mis lühendasid käikude arvu 22-ni.

Teine suur idee oli jätkata otsimist kui esimene lahendus oli leitud. Nagu oli välja toodud Thistlethwaite-i algoritmis, esimese faasi lahendusel võib olla veelgi parem teise faasi lahendus. Kui üks lahend on leitud, teised esimese faasi lahendused proovitakse samuti läbi otsimaks paremat teise faasi lahendust ning sellest tulenevalt paremat üleüldist lahendust. Kuid optimisatsiooni kohalt pole see veel kaugeltki lõpp. Võttes eelduseks, et lahend on 7+8 käiku ning kõik ülejäänud esimese faasi lahendid on läbi proovitud, võtame kasutusele IDA* otsingu esimese faasi jaoks, mis otsib lahendusi, mis on käikude arvult võrdne 8-ga, seejärel teise faasi lahendeid, mille käikude arv on ülimalt 6 (et summa 8+6 oleks väiksem kui see mis meil praegu on leitud (7+8)). Esimese faasi otsingu sügavuse kasvades väheneb teise faasi lahendite arv kuni esimene faas on saavutanud seni parima käikude arvu. Sellest tulenevalt võib lugeda lahendi optimaalseks.



```
maxLength=9999

function Kociemba ( position p)
  for depth d from 0 to maxLength
    Phase1search( p; d )
  endfor
endfunction

function Phase1search( position p; depth d )
  if d=0 then
    if subgoal reached and last move was a quarter turn of R, L, F, or B then
      Phase2start( p )
    endif
  elseif d>0 then
    if prune1[p]<=d then
      for each available move m
        Phase1search( result of m applied to p; d-1 )
      endfor
    endif
  endif
endfunction
```

```
function Phase2start ( position p)
  for depth d from 0 to maxLength - currentDepth
    Phase2search( p; d )
  endfor
endfunction

function Phase2search( position p; depth d )
  if d=0 then
    if solved then
      Found a solution!
      maxLength = currentDepth-1
    endif
  elseif d>0 then
    if prune2[p]<=d then
      for each available move m
        Phase2search( result of m applied to p; d-1 )
      endfor
    endif
  endif
endfunction
```

Joonis 1 sügavuse otsing

Lisatingimusena tuleb mainida, et esimene faas peab lõppema käiguga, mis ei kajastu teise faasi käikudes (F, F', B, B', R, R', L, L'). Kui see kitsendus jätta arvestamata siis kõik samad käiguahelad otsitakse uuesti kui esimese faasi maksimaalne käikude arv on suurendatud ühe võrra.

Algoritm iseenesest on ülimalt efektiivne. Algoritm leiab väga kiiresti lahendusi mis on ligilähedased optimaalsele käikude arvule. [9]

2.2.5. Korf-i algoritm

2.2.5.1. Heuristiline otsing

Viimatine kava tuletamaks heuristikat baseerub musterdandmebaasidel mis arendati Culbersoni ja Schaefferi eestvedamisel ning mida kasutati Korfi poolt leidmaks optimaalseid lahendusi Rubiku kuubikule. Sarnane probleem nagu 15-puslega, musterandmebaasi võib lugeda kui tabelit mis omab käikude arvu ning selle väärtusi abstraktse olekumudelina kus kindla seeria kuubikute asukoht ignoreeritakse. Kuna abstraktne olekumudel on oma suuruselt oluliselt väiksem originaalsest olekumudelist siis on võimalik rakendada n.ö. „pimeotsingut" (näiteks ulatus-esimesena otsing). Seega olek *s* heuristika *h(s)* suhtes saadakse võttes kauguse projektsioonist *s* sihiprojektsioonini *G* (goal) abstrakse olekumudeli sees. Kui leidub mitu sellist musterandmebaasi siis võetakse selle asemel kauguste maksimumväärtus. Musterandmebaaside idee on võimas, kuid mitte üldsegi tavapärane.

Korf ja Taylor visandasid teooria heuristika kohta: lükandpusle põhimõttel oli idee lahendada mingi arv abstraktseid (lõdvendatud) probleeme. Sellest tulenevalt saab igat alamprobleemi lahendada „toore jõu" meetodil. [10, p. 28]

2.2.5.2. Musterandmebaasid

On leitud, et keskmine käikude arv optimaalse lahenduskäigu saavutamiseks on 18. Algoritm mida kasutati on Iteratiivne-süvendamine-A (*iterative-deepening-A, IDA**) alampiirilise heuristilise funktsiooniga mis baseerub laial mälul-põhineval otsingutabelil ehk musterandmebaasil. Nimetatud tabelid hoiustavad täpset käikude arvu mida läheb vaja alametappide lahendamiseks, antud juhul siis liigutatavate kuubikute alamhulka (kusjuures iga alametapp on lahendatud siis kui kõik selle kuubiku 9 alamkuubikut on ühte värvi). [11]

Kuup segatakse kasutes suvalist arvu juhuslikke käike ning seejärel üritatakse taastada algne olek lahendatud kuubi näol. Võttes arvesse, et lahenduste arv igast suvaolekust on $4{,}3252*10^{19}$ on kuubi lahendamine üsna tülikas. Kui aga võtta abiks üldine strateegia mis koosneb makrooperatsioonidest, liigutamaks kindlalt positsioneeritud kuubi külgi nii, et juba eelnevalt positsioneeritud kuubi asendid ei muutuks. Sellised strateegiad võtavad üldiselt 50-100 käiku jõudmaks lahendatud kuubini, kuid on tõestatud, et igat suvaoleks kuupi on võimalik lahendada mitte rohkem kui 20 käiguga.

Probleemile lähenetakse leidmaks lühima käikude järjestuse mis lahendaks antud suvaolekus kuubi.

27st eksisteerivast 1x1x1 kuubikust 26 on nähtavad ning üks on keskmine peidetud kuubik. Nendest omakorda kuus kuubikut igal tahul on võimelised pöörama päri- ja vastupäeva, kuid mitte midagi enamat. Ülejäänud 20st kuubikust 8 on nurgakuubikud kus korraga on võimalik näha kolme tahku. Viimased 12 kuubikut on servakuubikud mis on seotud ainult kahe tahuga. Nurgakuubikud liiguvad vaid nurgapositsioonidel ning servakuubikud servapositsioonidel. Nurgakuubikut on võimalik positsioneerida kolmel erineval viisil, servakuubikut aga kahel erineval viisil. Sellest tulenevalt on võimalik unikaalselt spetsifitseerida kuubi olek andes positsiooni ja orientatsiooni kaheksa nurgakuubiku ja kaheteistkümne servakuubiku kohta. See on taasesitatav 20 elemendilise massiivina, üks iga kuubiku jaoks kus sisu kodeerub positsiooni ja orientatsiooni ümber üheks väärtuseks kahekümneneljast, 8x3 nurgakuubikute jaoks ning 12x2 servakuubikute jaoks. Sellest tulenevalt on tegelik võimaluste arv $8!*3^8*12!*2^{12}$. Edasised kitsendused vähendavad koguvõimaluste arvu 12 korda, sest terve probleemikogu koosneb kaheteistkümnest eraldi, kuid isomorfsest alamgraafist mis ei oma ühtegi lubatud käiku. Seega kõik maksimaalsed käigud taanduvad $8!*3^8*12!*2^{12}/12$-le mis õigupoolest võrdub 43 252 003 274 489 856 000 ga.

Depth	Nodes
1	18
2	243
3	3,240
4	43,254
5	577,368
6	7,706,988
7	102,876,480
8	1,373,243,544
9	18,330,699,168
10	244,686,773,808
11	3,266,193,870,720
12	43,598,688,377,184
13	581,975,750,199,168
14	7,768,485,393,179,328
15	103,697,388,221,736,960
16	1,384,201,395,738,071,424
17	18,476,969,736,848,122,368
18	246,639,261,965,462,754,048

Joonis 2 sõlmed otsipuus sügavuse funktsioonis

Järgmine lähenemine hõlmaks endas primitiivsete operaatorite defineerimist, milleks on kaks lähenemist: esimene lähenemine kirjeldab üht suvalise tahu 90 kraadilist pööret (veerandpööret). 180 kraadiline pööre kujutab endas kahte 90 kraadilist pööret samas suunas, nõnda võib väita, et primitiivseks pöördeks on suvatahu pööre kas 90 kraadi päri- või vastupäeva või 180 kraadiline pööre.

Loetledes suvalist kolme primitiivset pööret suvalise kuue tahu peal saame hargnemisteguriks 18. Võttes arvesse, et on ülearune pöörata sama tahku kaks korda järjest, saame pärast esimest pööret hargnemisteguriks 15. Lisaks sellele on vastastahkude pööramine sõltumatu ning kommutatiivne. Näiteks pöörates eesmist ning tagumist tahku viib sama tulemuseni kui kasutada vastassuunalist pööramist. Seega iga tahu vastaspaari jaoks valime järjekorra ning keelame käigud mis pööravad neid kahte tahku järjestikku vastupidises suunas. Siit saadud kitsendusega kärbime hargnemisteguri 15lt 13,34847-le.

Tavaliselt võetakse heuristilist lähenemist kui funktsiooni mis arvutatakse algoritmi poolt, seega igat funktsiooni võib arvutada ka otsingutabelina, andes piisavalt mälu. Efektiivsuse tõttu on heuristilised funktsioonid tavapäraselt ettearvutatud ning mällu talletatud. Näiteks *Manhattani* kauguse funktsioon on arvutatud tabeli abil mis omab iga 1x1x1 kuubiku *Manhattani* kaugust igast individuaalsest kuubikust igas võimalikus positsioonis ja orientatsioonis. Kui me võtame arvesse vaid kaheksa nurgakuubikut siis viimase kuubiku positsioon ja orientatsioon on kindlaks määratud ülejäänud seitsme kuubikuga, seega on vaid $8!*3^7$ = 88 179 840 võimalikku kombinatsiooni. Kasutades „ulatus-esimesena“ otsingut ning teades, et väärused esitavad end kujul nullist üheteistkümneni võime väita, et iga tabelisisendi jaoks läheb vaja nelja bitti. Sellest

tulenevalt vajab antud tabel 44 089 920 baiti mälu, või 42 megabaiti, mis tänapäeva mälu mõistes pole mingi kitsendus. Arvestada tuleks sellega, et antud tabelit tuleb arvutada korra iga alametapi lahendamiseks ja selle pikkust (käikude arvu) võib amortiseerida üle mitme samasuguse alametapi lahenduse. Nimetatud „musterandmebaaside“ tabelite kasutamine baseerub Culberson ja Schaefferil, 1996, kes rakendasid sama meetodit 15-puslel.

Eeldatav väärtus sellel heuristikal on ligikaudu 8,764 käiku, võrreldes 5,5 käiguga nurgakuubikute *Manhattani* kaugusega. IDA* otsingu käigus arvutatakse heuristilise tabeli unikaalne indeks iga oleku genereerimisel, seejärel ka viide tabelile. Talletatud väärtus on käikude arv mida läheb vaja nurgakuubikute lahendamiseks ning sellest tulenevalt alampiir terve kuubi lahendamiseks. Antud heuristikat on aga võimalik täiendada võttes arvesse ka servakuubikud. Kuue servakuubiku võimalike kombinatsioonide arv kaheteistkümnest on $12!/6!*2^6 = 42\ 577\ 920$. Võimalik käikude arv antud alametapi lahendamiseks on nullist kümneni, kus eeldatav käikude arv on umbes 7,668 käiku. Neli bitti iga kirje kohta antud tabelis nõuaks 21 288 960 baiti või 20 megabaiti. Tabel seitsme kuubiku jaoks nõuaks aga 244 megabaiti mälu. Ainus vastuvõetav viis kombineerimaks nurga- ja servaheuristikaid on võtta nende maksimum. Sarnaselt saab arvutada ka vastavad heuristilised tabelid ülejäänud kuue servakuubiku jaoks. Kogu vajaminev mälu summeerub 82 le megabaidile (42 megabaiti nurgakuubikute jaoks ning 2*20 megabaiti servakuubikute jaoks). Eeldatav käikude arv nende kolme heuristika korral on 8,878 käiku. Kuigi see kasv 8,764-lt 8,878-le on väike, võidame me rohkelt jõudluses. Saades rohkem mälu saaks arvutada ning talletada isegi suuremaid tabeleid.

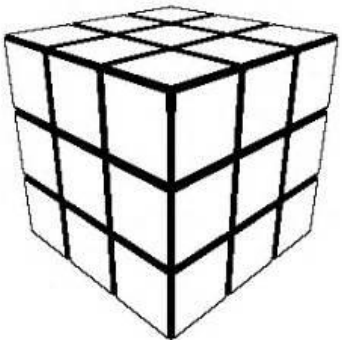
Jõudluse analüüs: järjepidev sõlmede arv mille genereerib IDA* suvalisel iteratsioonil üle erinevate probleemiinstantside viitab sellele, et algoritmi jõudlus on järeleandev analüüsile. [12]

3. Rubiku kuubiku rakenduse arenduskäik ja kasutajaliidese prototüüp

Antud peatükk hõlmab endas rakenduse arenduskäigu samm-sammulist kirjeldamist ning graafilise kasutajaliidese prototüüpi.

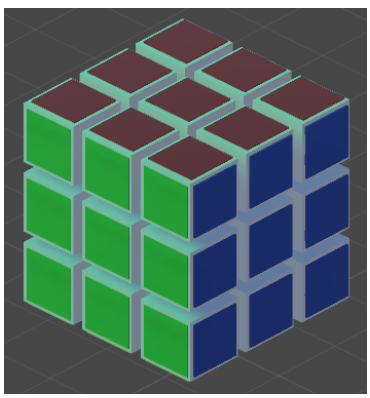
3.1 Rakenduse arenduskäik

Rakenduse loomise platvormiks valisin Microsoft Visual Studio Community 2017 ning Unity 3D. Programmid sai eelkõige valitud selle pärast, et on olemas varasem kasutamiskogemus nende programmidega. Programne pool Visual Studios on kirjutatud C# keeles ning haakub 3D mudeliga Unity programmi külge mis representeerib visuaalset poolt. Esimesena sai üles ehitatud 27 erinevat kuubikut ning seejärel üles seatud kolmes kihis 3x3x3 tasandil.



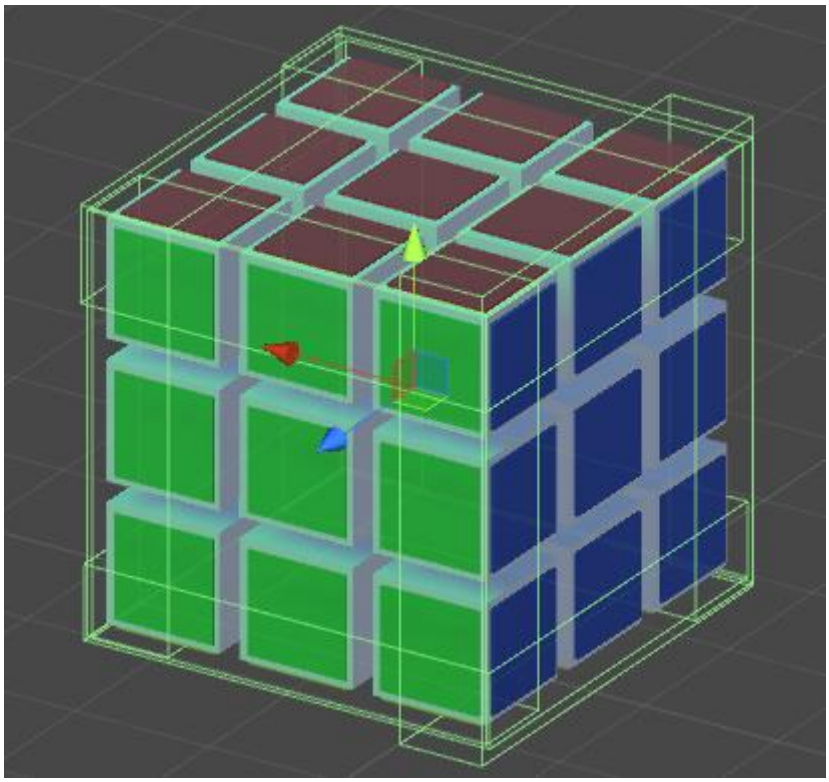
Joonis 3 – esialgne baaskujund kuubi arendamiseks.

Järgmiseks sai ära defineeritud põhitahud mida kasutan kuubi lahendamisel (U, D, L, R, F, B). Kuubikud sai omavahel ära seotud tasandiliselt selliselt, et iga vastav tähistus liigutab kõiki kuubikuid sellel tasandil. Paremaks visualiseerimiseks sai ka värvid külge lisatud, et oleks reaalselt näha kuubi tahkude liigutamise tulemusi.



Joonis 4 – värvidega kaetud kuup.

Pärast värvide lisamist sai omavahel ühendatud 1x1x1 tahud *collider*-ide abil.



Joonis 5 – kõik kuus, U, D, L, R, F ja B *collider*-it

Kui põhilised tähistused olid defineeritud, sai lisatud GUI kus saab valida manuaalse ning algoritmilise lahendamise vahel. Manuaalne lahendamine toob ette 6x2 nupumenüü (põhitähistus ning selle inversioonid). 180 kraadilist pööret loetakse antud juhul üheks käiguks, seega eraldi 180 kraadi pööramise nuppu ei kasutata.

Manuaalsel lahendamisel on n.ö. *freeroam* funktsioon mis laseb kuupi vabalt keerata ning iga nurga alt vaadata.

Automaatsel lahendamisel on visuaalne liugur millega on võimalik modifitseerida visuaalset lahendamiskiirust.

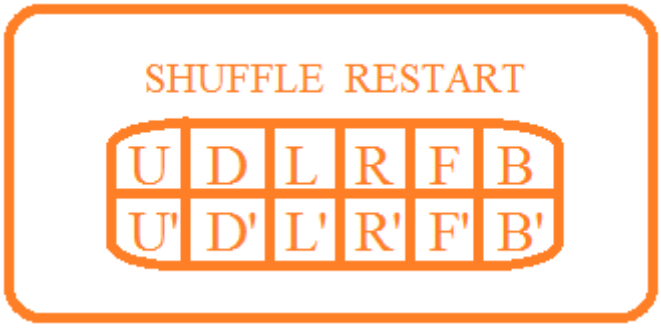
3.2 Kasutajaliidese prototüüp

Luues prototüübile kasutajaliidest valisin põhifunktsioone kirjeldavad nupud mida prototüüp lahendama peaks:

Shuffle: sooritab juhusliku arvu juhuslikke tahupöördeid, miinimum 20.

Restart: viib kuubi viimasesse asendisse mida enne manuaalset sisestamist kasutati. Erineb *shuffle*-i poolest selle võrra, et kuupi ei segata uuesti vaid taastatakse käigud mis olid eelnevalt kasutaja poolt tehtud kasutades paneelil olevaid nuppe:

U – up, D – down, L – left, R – right, F – front, B – back.



Joonis 6 – GUI visuaalne interpretatsioon

Kokkuvõte

Olemasoleva bakalaureusetöö eesmärgiks oli Rubik-u kuubiku meetodite, algoritmide ja lahenduskäikude kirjeldamine läbi detailse analüüsi ning kirjelduse. Kõrvaleesmärk oli esitada projekti kirjeldusele loodud programne kasutajaliidese prototüüp Unity 3D ja Microsoft Visual Studio 2017 programmiga.

Rubik-u kuubiku programmi luues tõin välja rakenduse tööetapid, üldkirjelduse, pildid prototüübist ja GUI-st. Lisaks kirjeldasin rakenduse funktsionaalsust ning kasutajaliidese samm-sammulist arendus- ja kasutuskäiku.

Esimeses peatükis ilmnes, milliseid meetodeid kirjeldama hakatakse ning millised saavad üldised meetodid olema.

Mahukaimas teises peatükis anti lugejale mõista millised on erinevad algoritmid ning kuidas neid saab rakendada Rubik-u kuubi lahendamisel

Kasutatud materjalid

[1] „Manhattan distance,“ [Võrgumaterjal]. Available: https://qph.ec.quoracdn.net/main-qimg-4f3038d129083c95e9ba7231bec36747. [Kasutatud 21 Mai 2017].

[2] D. Ferenc, „Ruwix Advanced CFOP Fridrich,“ [Võrgumaterjal]. Available: https://ruwix.com/the-rubiks-cube/advanced-cfop-fridrich/. [Kasutatud 22 Mai 2017].

[3] J. Jelinek, „Ortega Corners-First Method,“ 2000-2011. [Võrgumaterjal]. Available: http://www.cs.brandeis.edu/~storer/JimPuzzles/RUBIK/Rubik3x3x3/SOLUTIONS/Rubik3x3x3SolutionOrtegaAndJelinek.pdf. [Kasutatud 22 Mai 2017].

[4] G. R. Schultz, „CCL algorithms (3x3x3),“ [Võrgumaterjal]. Available: https://www.speedsolving.com/wiki/index.php/CLL_algorithms_(3x3x3). [Kasutatud 22 Mai 2017].

[5] M. Waterman, „Waterman method - speedcubing,“ [Võrgumaterjal]. Available: https://www.speedsolving.com/wiki/index.php/Waterman_Method. [Kasutatud 10 Mai 2017].

[6] H. K. M. D. J. D. Tomas Rokicki, „God's number is 20,“ [Võrgumaterjal]. Available: cube20.org. [Kasutatud 17 Mai 2017].

[7] M. B. Thistlethwaite, „Thistlethwaite's 52-move algorithm,“ [Võrgumaterjal]. Available: https://www.jaapsch.net/puzzles/thistle.htm. [Kasutatud 10 Mai 2017].

[8] D. Dhondiyal, „IDA-Star (IDA*) Algorithm in general,“ [Võrgumaterjal]. Available: https://algorithmsinsight.wordpress.com/graph-theory-2/ida-star-algorithm-in-general/. [Kasutatud 18 Mai 2017].

[9] J. Scherphuis, „Computer puzzling,“ [Võrgumaterjal]. Available: https://www.jaapsch.net/puzzles/compcube.htm#kocal . [Kasutatud 21 Mai 2017].

[10] B. Bonet ja H. Geffner, „Planning as heuristic search,“ [Võrgumaterjal]. Available: http://www.sciencedirect.com/science/article/pii/S0004370201001084 . [Kasutatud 21 Mai 2017].

[11] A. F. Richard E. Korf, „Disjoint pattern database heuristics,“ [Võrgumaterjal]. Available: http://www.sciencedirect.com/science/article/pii/S0004370201000923. [Kasutatud 18 Mai 2017].

[12] R. E. Korf, „1997 - Finding optimal solutions to Rubik's Cube using pattern databases,“ [Võrgumaterjal]. Available: https://www.aaai.org/Papers/AAAI/1997/AAAI97-109.pdf. [Kasutatud 15 Mai 2017].