TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Mirjam Pajumägi 164004IAPB

MÄNGU TENPAIR LAHENDUSSTRATEEGIATE ANALÜÜS

Bakalaureusetöö

Juhendaja: Marko Kääramees
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: Mirjam Pajumägi

21.05.2019

Annotatsioon

Käesoleva lõputöö eesmärgiks on analüüsida erinevate algoritmide baasil mängu TenPair lahendatavust ning luua nende põhjal strateegiaid, mis lahendamise optimaalsust parandaks. Samuti kirjeldada, kuidas erinevad strateegiad ning algoritmid toimivad ning milliseid tulemusi on võimalik neid kasutades saavutada.

Antud eesmärkide saavutamiseks viidi läbi katsed enda poolt implementeeritud mängul mitmete erinevate otsingustrateegiatega, hinnati nende efektiivsust ning optimaalsust läbi vastavalt strateegiatele seatud piirangute. Vaatluse all olid *Brute Force*, A* ning *Monte Carlo Tree Search* algoritmid. Kõiki algoritme nii originaallaua kui ka erineva suurusega juhuslike mängulaudade peal.

Selgus, et originaalmängulauda pole võimalik *Brute Force* meetoditega lahendada, kuid seda on võimalik teha *Monte Carlo Tree Search*, A* ning *Best First* strateegiatega. Juhuslikest suurematest mängulaudadest andsid kõige optimaalsemad tulemused *Best First* ja A* strateegiad.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 27 leheküljel, 5 peatükki, 21 joonist, 0 tabelit.

Abstract

Analysis of Solving Strategies for Game TenPair

There are a plethora of games in the world that are based on math and logic. A popular example from recent times would be the game "2048". There has been much research to find out and analyse the best strategies to solve these games with optimal time and/or memory consumption.

The game "TenPair" is also a game based on math and logic. It is played on a grid such as notebook paper and the goal is to remove all numbers from the game by crossing out matching pairs. "Tenpair" can be played both online or manually on paper. However, due to its lack of popularity, there has been no(/little) research about the strategies for this game.

The aim of this thesis is to analyse different algorithms and solving strategies in order to find the most optimal method for solving the game. An additional goal is to describe how different methods work and what sorts of results they are able to produce.

To achieve this, the game itself was implemented with variable board widths and matching rules and then tested with various solving strategies to find the most optimal way to solve the game. The four main algorithms that this thesis was based on were *Brute Force, Monte Carlo Tree Search, Best First Search*, and A*. All of these were tested on the traditional starting board, as well as random boards of various sizes and matching rules.

It turned out that the original board is impossible to solve with the basic *Brute Force* algorithm, because the memory consumption gets too big. However, it is possible to solve the main board with either *Monte Carlo Tree Search*, A*, or *Best First* strategies. Bigger random boards were solved the fastest by the *Best First* strategy.

The thesis is in Estonian and contains 27 pages of text, 5 chapters, 21 figures, 0 tables.

Lühendite ja mõistete sõnastik

Best First	*Best First Search*.
Branching factor ehk hargnevustegur	Järglaste arv otsingupuus.
Brute Force	Algoritm, mis otsingupuus kontrollib kõiki järglasi, kuni jõuab lahendini.
Heuristika	Meetod või reegel probleemi lahendamiseks. [1]
MCTS	*Monte Carlo Tree Search*.
MCTS *backpropagation*	*Monte Carlo Tree Search*'i etapp, mis järgneb *simulation*'le. Liikudes tagasi otsingupuu juurelementi, uuendatakse kõiki temast allpool olevaid järglasi vastavalt *simulation* etapis saadud tulemustele.
MCTS *expansion*	*Monte Carlo Tree Search*'i etapp, mis järgneb *selection*'le. Luuakse otsingupuus valitud järglasele järgmised järglased.
MCTS *selection*	*Monte Carlo Tree Search*'i esimene etapp, kus otsingupuus valitakse järglastest kõige parema hinnanguga järglane (*leaf node*).
MCTS *simulation*	*Monte Carlo Tree Search*'i etapp, mis järgneb *expansion*'le. Valitakse juhuslikult loodud järglastest üks ning simuleeritakse temaga järgmised käigud, kuni jõutakse kas võidu või kaotuseni.
Memoization	Optimeerimismeetod, mille kaudu vähendatakse korduvaid otsinguid samal andmehulgal, tagastades vahemällu salvestatud andmehulga. [2]
Reinforcement learning ehk stiimulõpe	Tarkvara agent, mille eesmärk on teostada tegevusi ümbritsevas keskkonnas nii, et tegevustest tulenev preemia oleks maksimaalne. [3]

Sisukord

Jooniste loetelu

1 Sissejuhatus

Matemaatilisi mänge on mitmeid, nendest ilmselt kõige tuntum on mäng nimega “2048”, mille lahendamiseks mõeldud strateegiaid on palju uuritud. TenPair on samuti matemaatiline loogikamäng, kuid selle lahendamiseks pole tehtud uurimistöid ega testitud selle lahendatavust erinevate algoritmidega.

Mängu TenPair puhul on tegemist loogikamänguga, mille eesmärk on saada mängulaud kõikidest arvudest tühjaks. Arve saab paarikaupa eemaldada kindla reeglistiku järgi:

- Arvupaaris peavad mõlemad arvud olema samad (nt 5 ja 5) või peab nende arvude summa olema 10 (nt 2 ja 8).
- Arvud peavad olema järjestikku/kõrvuti kas vertikaalselt või horisontaalselt, kusjuures horisontaalselt võib arve eemaldada ka juhul, kui üks neist on rea lõpus ja teine järgmise rea alguses, st üle rea.
- Arvude eemaldamisest jäävad tühjad ruudud järjestikkust ei mõjuta.

Mängulauda on võimalik duplitseerida. Seda saab erinevates mängu versioonides teha kas igal ajal nö käiguna või juhul, kui kõik paarid on mängulaualt eemaldatud. Duplitseerimise käigus kirjutatakse mängulauale jäänud arvud mängulaua lõppu, kusjuures tühje ruute ei lisata. See annab võimaluse mängu jätkata ning uusi paare leida. Antud töös kasutati reeglistikku, kus lauda võib duplitseerida vaid juhul, kui kõik sobivad paarid on mängulaualt eemaldatud.

Standardse mängu alglaud koosneb 9 kaupa järjestikku kirjutatud numbritest 1-19 (v.a. 10), nagu toodud Joonis 1. Keerulisemates versioonides võib mängu alglaud olla genereeritud juhuslikult.

1	2	3	4	5	6	7	8	9
1	1	1	2	1	3	1	4	1
5	1	6	1	7	1	8	1	9

Joonis 1. Standardse mängulaua algseis

Mängulaua algseisust on maksimaalselt võimalik eemaldada viis paari (1-1, 1-1, 1-9, 1-9, 2-8). Tekkiv mängulaud on välja toodud Joonis 2.

	2	3	4	5	6	7		
				1	3	1	4	
5	1	6	1	7	1	8		

Joonis 2. Mängulaua seis peale kõikvõimalike paaride eemaldamist

Kuna mängulaual pole rohkem paare, mida eemaldada, tuleb laud duplitseerida. Kuna antud juhul jättis alumises reas eemaldatud paar 1-9 tühjad kohad, mida täita ei tohi, algab juurde lisatud mängulaud uuelt realt. Duplitseeritud mängulaud on toodud Joonis 3.

	2	3	4	5	6	7		
				1	3	1	4	
5	1	6	1	7	1	8		
2	3	4	5	6	7	1	3	1
4	5	1	6	1	7	1	8	

Joonis 3. Duplitseeritud mängulaud

Antud töö eesmärgiks on analüüsida erinevate juba olemasolevate algoritmide baasil mängu lahendatavust ning võimaluse korral luua alus ka mängu ise lahendavale tehisintellektile. Samuti on eesmärk kirjeldada, kuidas erinevad strateegiad ning algoritmid toimivad ning milliseid tulemusi on võimalik neid kasutades saavutada.

Antud eesmärkide saavutamiseks viiakse läbi katsed enda poolt implementeeritud mängul mitmete erinevate otsingustrateegiatega, hinnatakse nende efektiivsust ning optimaalsust läbi vastavalt strateegiatele seatud piirangute. Eesmärk on tuvastada, millised parameetrid erinevate meetodite ja algoritmide puhul määravaks saavad, et leida kõige optimaalsem lahendus võimalikult vähese aja- ning mälukasutusega.

Antud töö on jagatud peatükkideks ja alampeatükkideks. Teises peatükis antakse ülevaade hiljem töös rakendatud algoritmidest. Kolmas peatükk on jagatud viieks alampeatükiks ning nendes analüüsitakse valitud algoritmide ja strateegiate tööd juhuslikel väiksematel mängulaudadel, originaalmängulaual ning juhuslikel suurematel mängulaudadel. Lisaks võrreldakse detailsemalt tulemusi lähtudes suuremate mängulaudade lahenduvusest.

2 Ülevaade töös kasutatavatest meetoditest

Mängule sobiva lahendusstrateegia leidmiseks uuriti erinevate matemaatiliste mängude jaoks kasutatud lahendusalgoritmide ning nende strateegiate kohta, mida töös hiljem kasutati.

2.1 *Brute Force*

Brute Force otsing on algoritm, mille eesmärk on leida lahendus probleemile läbi kõikvõimalike variantide lineaarselt läbi otsimise. Tegu on väga lihtsasti implementeeritava algoritmiga, mis leiab alati lahenduse, kui see eksisteerib, kuid mis on piiratud probleemi keerukuse ning püstitusega. Seetõttu mida keerulisem on probleem ning mida rohkem on võimalikke väljundeid, seda aja- ja/või mälukulukamaks läheb algoritm. Seega sobib see limiteeritud suurustega probleemide lahendamiseks. Samuti kasutatakse algoritmi võrdluseks teiste algoritmide vastu [1].

Antud töös kasutati *Brute Force* algoritmi laiuti otsinguna, et teha kindlaks, kas mängu on võimalik lahendada triviaalselt, st kõikvõimalikke lauaseise läbi proovides. Samuti kasutati strateegiat saamaks aimu, kui keerulise mänguga on tegu ning millised on mängu keerukust mõjutavad parameetrid.

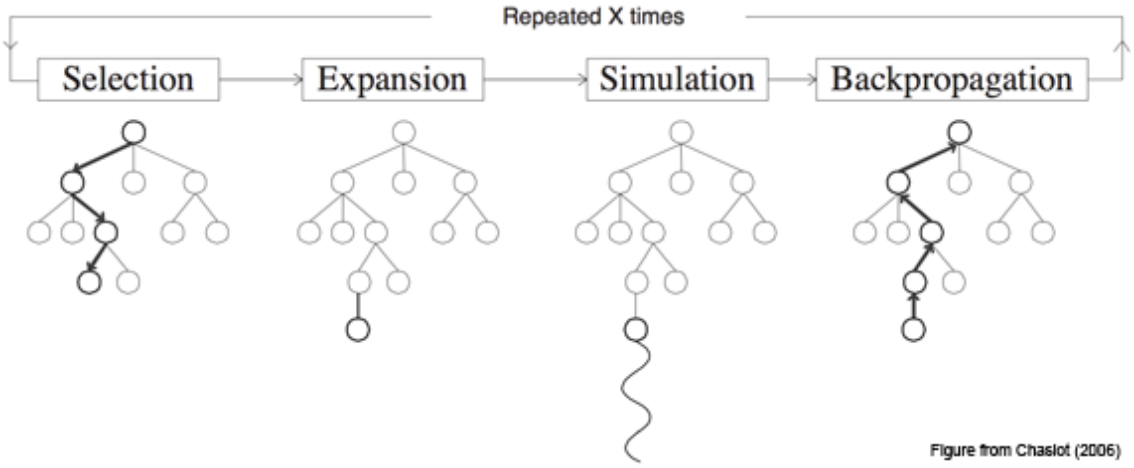
2.2 *Monte Carlo Tree Search*

TenPair'i puhul on tegemist mänguga, mille hargnevustegur on eksponentsiaalselt kasvav, st igas mänguseisust on keskmiselt võimalik teha suhteliselt palju käike. See tõstab kõikide variantide läbitöötamiseks vajamineva arvutusressurssi väga kiirelt väga kõrgeks.

Selleks, et lahendada näiteks MinMax algoritmil esinevat olukorda, kus parima võimaliku käigu otsinguil kaalutakse läbi kõikvõimalikud käigud, mis on piiratud ajaga suhteliselt võimatu, on loodud sellele ligilähedane algoritm *Monte Carlo Tree Search* (edaspidi MCTS), mis kasutab tõenäosuslikku otsingut.

MCTS puhul on osad käigud tähtsamad või parema kaaluga kui teised. Sellised käigud on tõenäolisemalt need, mis viivad kiiremini soovitud tulemuseni. Mängu TenPair jaoks on soovitud tulemus tühi laud, kust kõik numbrid on eemaldatud. MCTS leiab otsingupuus seisu, millel on kõige suurem tõenäosus mängu võita (*selection*). Juhul kui seis ei ole soovitud tulemus, antud töös mängu lõpp, luuakse sellele seisule järgnevad võimalikud seisud (*expansion*). Algoritm kasutab *reinforcement learning* lähenemist, et valida nendest parim ning arvutab, kui lähedal oleks algne seis peale käiku soovitud tulemusele. Käik saab endale hinnangu (*simulation*). Parima hinnanguga seis võetakse arvesse ning sellele vastavalt uuendatakse mänguseisude võidutõenäosusi eelnevatel mänguseisudel (*backpropagation*) [2], [3].

Algoritmi tööd illustreerib Joonis 4 ning pseudokood on välja toodud Joonis 5.



Joonis 4. *Monte Carlo Tree Search* algoritmi tööd illustreeriv joonis [3]

Kõike seda tehakse niikaua, kuni leitakse käik, mis võidab ning lõpetab mängu. Otsingupuust parima võimaliku seisu leidmiseks kuluvat aega saab piirata, st mida pikem on valiku tegemise aeg, seda kaugemale jõuab algoritm otsingupuus vaadata ning seda tõenäolisemalt leiab soovitud tulemusele ligilähedasi mänguseise või isegi lõppseisu.

MCTS on olnud kasutuses nii Ruubiku kuubiku lahendamisel kui ka keerulisemate mängude nagu Go, male ja shogi lahendamiseks. Kuna tegu on suhteliselt üldise algoritmiga, on võimalik, et sellega saaks lahendada kõik kolm eelpool nimetatud keerulist mängu [5].

```
# main function for the Monte Carlo Tree Search
def monte_carlo_tree_search(root):

    while resources_left(time, computational power):
        leaf = traverse(root)
        simulation_result = rollout(leaf)
        backpropagate(leaf, simulation_result)

    return best_child(root)

# function for node traversal
def traverse(node):
    while fully_expanded(node):
        node = best_uct(node)

    # in case no children are present / node is terminal
    return pick_univisted(node.children) or node

# function for the result of the simulation
def rollout(node):
    while non_terminal(node):
        node = rollout_policy(node)
    return result(node)

# function for randomly selecting a child node
def rollout_policy(node):
    return pick_random(node.children)

# function for backpropagation
def backpropagate(node, result):
    if is_root(node) return
    node.stats = update_stats(node, result)
    backpropagate(node.parent)

# function for selecting the best child
# node with highest number of visits
def best_child(node):
    pick child with highest number of visits
```

Joonis 5. *Monte Carlo Tree Search* pseudokood [4].

Antud töös on MCTS implementeerimiseks kasutatud Paul Sinclairi poolt loodud *Monte Carlo Tree Search* paketti, mis on loodud programmeerimiskeel Pythoni jaoks. Seda saab kasutada olukordades, mille tulemid on täielikult vaadeldavad ning selle kasutamiseks tuleb paika panna vaid mänguspetsiifilised reeglid [6].

Otsingus eelistatakse järgmise sammu valikut tehes pigem läbi mängimata järglasi kui juba läbi mängitud häid järglasi, kuna puudub muu lõppseisu hinnang kui võit vs kaotus.

2.3 A*

A* algoritmi kõige silmapaistvam omadus on oskus kasutada hinnangu arvutamiseks nii juba läbitud teekonda kui eeldatavat teekonda, mis viib lõpuks sihini. Juba läbitud teekonda märgitakse tavaliselt g ning see on kõige lihtsamas mõttes kaugus alguspunktist. H ehk eeldatav kaugus lõppolekust on arvutatud heuristiliselt. Nende summa (Valem (1)) annab koguhinnangu, mis on ühtlasi ka iga mänguseisu koguhinnang. Hinnangud arvutatakse uuesti igal käigul [7].

$$f(n) = g(n) + h(n) \quad (1)$$

Funktsioon *f(n)* määrab minimaalse pikkuse, millest ükski antud otsingusõlme läbiv mäng ei saa olla lühem.

Uue käigu tegemisel valitakse järgnevad mänguseisud nende hinnangu alusel. Valituks saab mänguseis, mille kaudu on parim võimalus jõuda võimalikult kiiresti lõpp-punktini. Kui näiteks laiuti otsing ja Dijkstra algoritm vaatavad läbi kõik järgnevad mänguseisud, prioritiseerimata ühtegi võimalust, siis A* prioritiseerib neid, mis on koguhinnangult kõige lähemal võidule.

A* pseudokood on toodud Joonis 6.

```
// A* (star) Pathfinding
// Initialize both open and closed list
let the openList equal empty list of nodes
let the closedList equal empty list of nodes
// Add the start node
put the startNode on the openList (leave it's f at zero)
// Loop until you find the end
while the openList is not empty
    // Get the current node
    let the currentNode equal the node with the least f value
    remove the currentNode from the openList
    add the currentNode to the closedList
    // Found the goal
    if currentNode is the goal
        Congratz! You've found the end! Backtrack to get path
    // Generate children
    let the children of the currentNode equal the adjacent nodes
    for each child in the children
        // Child is on the closedList
        if child is in the closedList
            continue to beginning of for loop
        // Create the f, g, and h values
        child.g = currentNode.g + distance between child and current
        child.h = distance from child to end
        child.f = child.g + child.h
        // Child is already in openList
        if child.position is in the openList's nodes positions
            if the child.g is higher than the openList node's g
                continue to beginning of for loop
        // Add the child to the openList
        add the child to the openList
```

Joonis 6. A* algoritmi pseudokood [8]

Antud töös on A* implementeerimiseks kasutatud Julien Riallandi poolt loodud A* algoritmi implementatsiooni, mis on sarnaselt MCTS'le loodud Pythonis. Selle kasutamiseks tuleb realiseerida vaid soovitud hinnangud ning mängureeglite alusel paika panna eesmärk, kuhu soovitakse jõuda [9].

3 Tulemused

Analüüsi teostamiseks implementeeriti Pythonis kolm algoritmi koos strateegiatega ning katsetati neid nii originaallauaga kui erinevate piirangutega.

3.1 Algne analüüs *Brute Force* algoritmiga

Selleks, et saada aimu mängu keerukusest ja seda mõjutavatest parameetritest, implementeeriti mängu reeglitele *Brute Force* lahendus. Mängulaudade salvestamisel kasutati *memoization* tehnikat, st kui mõnda mänguseisu saab mööda otsingupuud mitmest kohast, siis mänguseisu topelt ei salvestata. Iga mängulauda esineb otsingupuus täpselt 1 korra.

Mängulaud salvestati *string* andmetüüpi, kus reavahetusi pole. Numbrid on numbrid ning tühjad/kustutatud kohad on märgitud 0'ga. Kasutatakse *intern* funktsiooni, et garanteerida, et samasugused mängulauad oleks mälus salvestatud sama objektina. Seda funktsiooni rakendati kõikidele mängulaudadele.

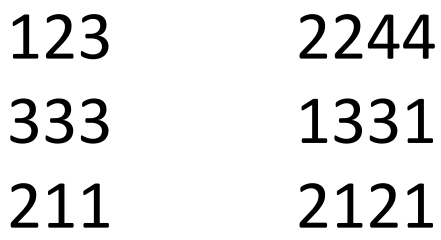
Selgus, et antud mängu originaallauale lahenduse leidmine triviaalse *Brute Force* algoritmiga pole mõistliku aja jooksul võimalik, kuna otsinguruum läheb liialt suureks.

Seejärel seati mõningad lisapiirangud, et testida, millistest muutujatest otsingu keerukus sõltub.

3.2 Juhuslike väiksemate mängulaudade lahenduvus

Esimeseks piiranguks seati mängulaua suurus. Selleks kohandati klassikalise mängulaua algseisu väiksematele kujudele. See tähendab, et kui algne mängulaud suurusega 9 koosnes arvudest 1-9, siis väiksemad lauad koosnevad vastavalt 3: 1-3, 4: 1-4 jne. Ühe mängulaua reas olevate numbrite arv oli võrdne mängulaua suuruse tähisega ning ridu oli alati 3. Samuti muudeti kustutamiseks sobivate arvude summa reeglit. See oli alati

ühe võrra suurem, kui mängulaua suurus. Seega näiteks mängulaua suurusega 3 sobiv summa reegel oli 4. Näitelauad on toodud Joonis 7.



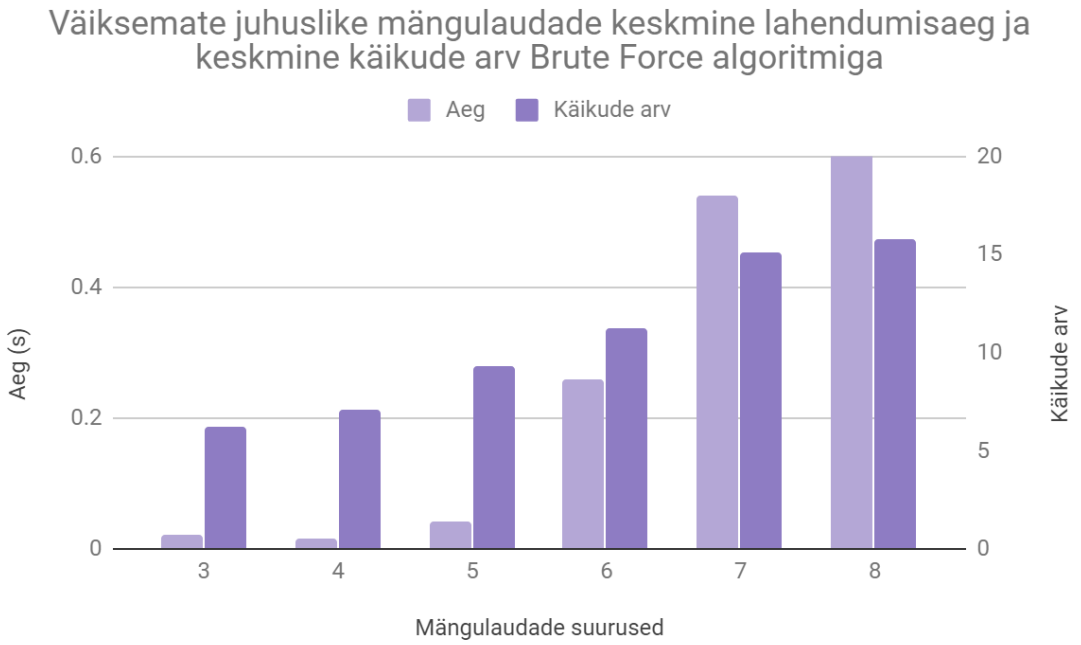
Joonis 7. Juhuslikud näitemängulauad 3 ja 4 lauasuuruse korral

Algoritmi töö hindamiseks viidi läbi 20 katset juhuslikku järjekorda aetud väiksemate laudade peal, eesmärgiga selgitada välja erinevate algoritmide poolt lahenduse leidmise kiirus ning selleni viinud sammude arv. Kõikide strateegiate puhul kasutati samu 20 lauda.

3.2.1 *Brute Force* väiksematel mängulaudadel

Esimesena viidi suvalise 20 väiksema laua peal läbi katse *Brute Force* algoritmiga, mis oli implementeeritud laiuti otsinguna. Algoritmile seati piiranguks maksimaalne lubatud lauasuurus 100 numbrit ning maksimaalne lubatud käikude arv, mis oli samuti 100.

Uuriti välja, kas algoritm suudab leida lahenduse mängulauale ning kui jah, siis kui kiiresti. Iga mängulaua suuruse kohta arvutati keskmine lahenduskiirus ning keskmine lahenduseni viinud käikude arv. Saadud tulemusi illustreerib Joonis 8.



Joonis 8. *Brute Force* algoritmiga väiksemate mängulaudade lahenduvus

Selgus, et mida suuremaks läheb mängulaud, seda keerulisem on laiuti otsinguga leida sobivat lahendit. Lahendi leidmise aeg tegi eriti hüppelise kasvu suurusega 8 laudade juures, kus keskmine aeg oli 3,77 sekundit. Joonisel on see võrdluse lihtsuse huvides märgitud kui > 0,6 sekundit.

Käikude arv muutus enam-vähem lineaarselt sõltuvalt laua suurusest. Kõik juhuslikult genereeritud lauad olid lahenduvad.

3.2.2 *Monte Carlo Tree Search* väiksematel mängulaudadel

Monte Carlo algoritmi otsingu piiranguks olid kasutatud koodipaketil esialgu otsingusügavuse aeg ehk simuleerimise maksimaalne aeg.

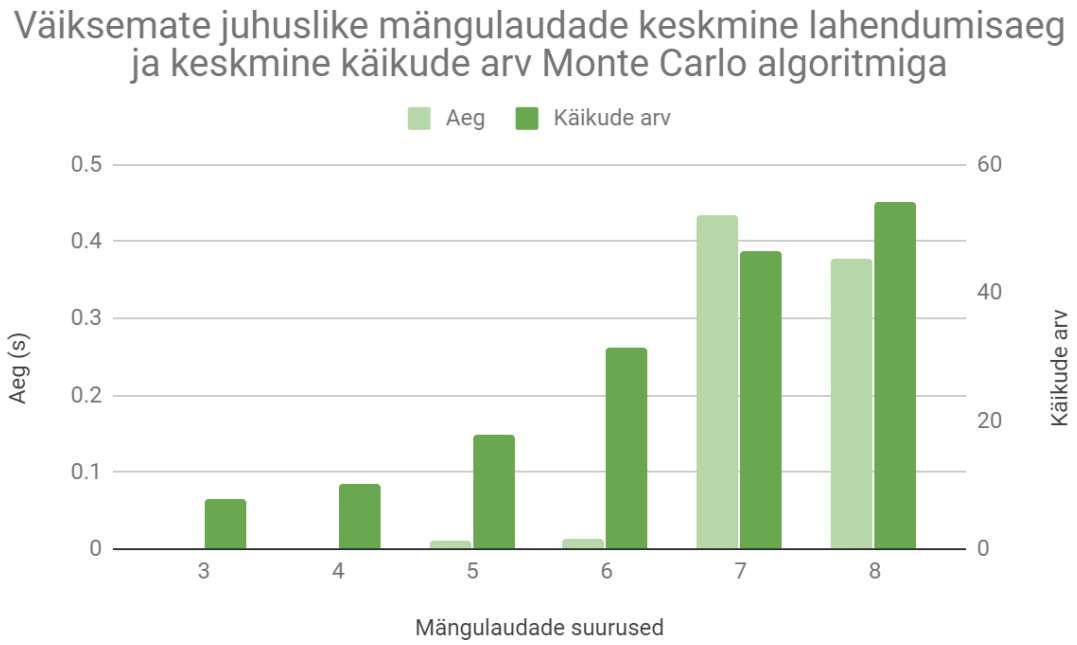
Selleks, et strateegia jõuaks alati lahenduseni, modifitseeriti töös kasutatud MCTS paketti nii, et lahenduse otsimine ei oleks piiratud aja või laua suurusega, vaid otsingupuus liigutakse nii kaua, kuni jõutakse sobiva lahendini.

Selleks hinnati võidulaua väärtust 1 000 000 000-ni ning kõik ülejäänud vahepealsed mängulauad said hinnanguks 0. Antud võidulaua hinnang sai valitud nii suur pigem harjumusest, kui algoritmi vajadusest. Kuna antud töös kasutatakse modifitseeritud

algoritmi, mis otsib nii kaua, kuni jõuab lahendini, pole tehniliselt oluline, kui suur on võiduseisu hinnang. Algoritmist lähtuvalt peab see siiski olema suurem või võrdne 1'ga. Sellest hoolimata annaks töös kasutatud suurem hinnang võimaluse hinnata ka vahepealseid tulemusi, vältimaks olukorda, kus lõpetamata mäng on suurema väärtusega kui lahenduse leidnud mäng.

*Monte Carlo Tree Search*i eripäradest lähtuvalt on iga sama alglauaga tehtud mäng erinevate tõenäosushinnangutega ning seetõttu ka lahenduseni jõudev teekond erinev.

Monte Carlo strateegiat rakendati samadel juhuslikel laudadel, mis olid kasutuses *Brute Force* puhul. Igale lauale rakendati otsingut 20 korda ning võeti nende ajalised keskmised ning käikude arvude keskmised. Sama suurusega laudade tulemustest leiti omakorda aja ning käikude hulga keskmised. Tulemused on näha Joonis 9.



Joonis 9. Monte Carlo algoritmiga väiksemate mängulaudade lahenduvus

Joonis 9 on näha, et lahenduse leidmiseks kulunud aeg ning käikude arv on omavahel üldiselt võrdelises seoses. Mängulaua algseisu suurenedes kasvab ka aeg ning käikude arv, seda eriti suuremate laudade puhul. Väiksemad mängulauad (3-6) lahendas strateegia <0,1 sekundiga, suuremate laudade puhul (7-8) ulatus aeg keskmiselt kuni 0,4 sekundini.

Tehtud käikude pealt on näha, et käikude arv MCTS'il on keskmiselt oluliselt suurem, kui *Brute Force* lahendustel samadel mängulaudadel.

Kõik mängulauad olid antud strateegiaga lahenduvad.

3.2.3 A* Search ja *Best First* väiksematel mängulaudadel

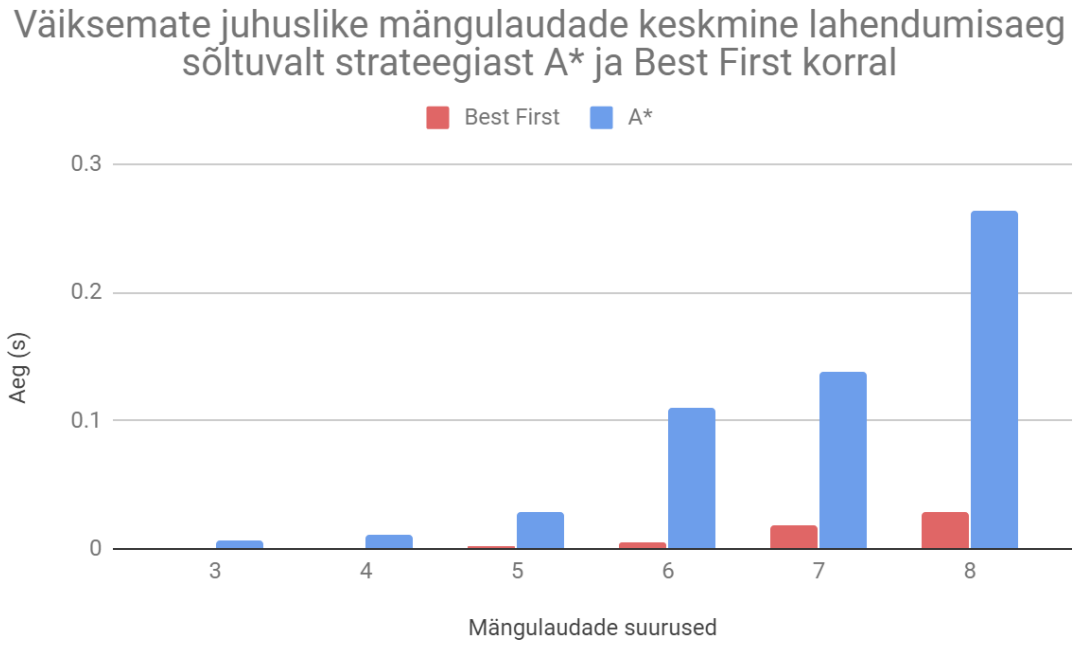
Kuna otsinguruum kasvab väga kiiresti, siis oli vaja otsida targemalt. Kõige lihtsam on otsida eelisjärjekorras edasi nendest seisudest, kus laua suurus on kõige väiksem. Selleks kasutati heuristilist funktsiooni.

Heuristiline funktsioon $h(n)$ määrati laua suurusega – mängulaual olevate sümbolite arv jagatud kahega, st teoreetiliselt kõige väiksem käikude arv, mida peab tegema, et mängulaud puhtaks saada.

Esimene katsetatud algoritm oli klassikaline A*. Kuna otsitakse kõige lühemat lahendust, siis funktsioon $g(n)$ on sooritatud käikude arv. Selliselt leiab A* antud algseisust kõige lühema võimaliku võiduka mängu

Lisaks katsetati ka A* erijuhtumit, kus $g(n)$ on alati võrdne nulliga. Seda algoritmi nimetatakse *Best First Search* (edaspidi *Best First*). See algoritm kasutab ainult lauasuurust, et valida, kust otsingut jätkata, seega otsib alati edasi kõige lühemast lauast. Kuna tundus, et pole põhjust, miks väiksema käikude arvuga lauaseis on parem, kui suurema käikude arvuga, siis tundus antud algoritm loogiline katsetada.

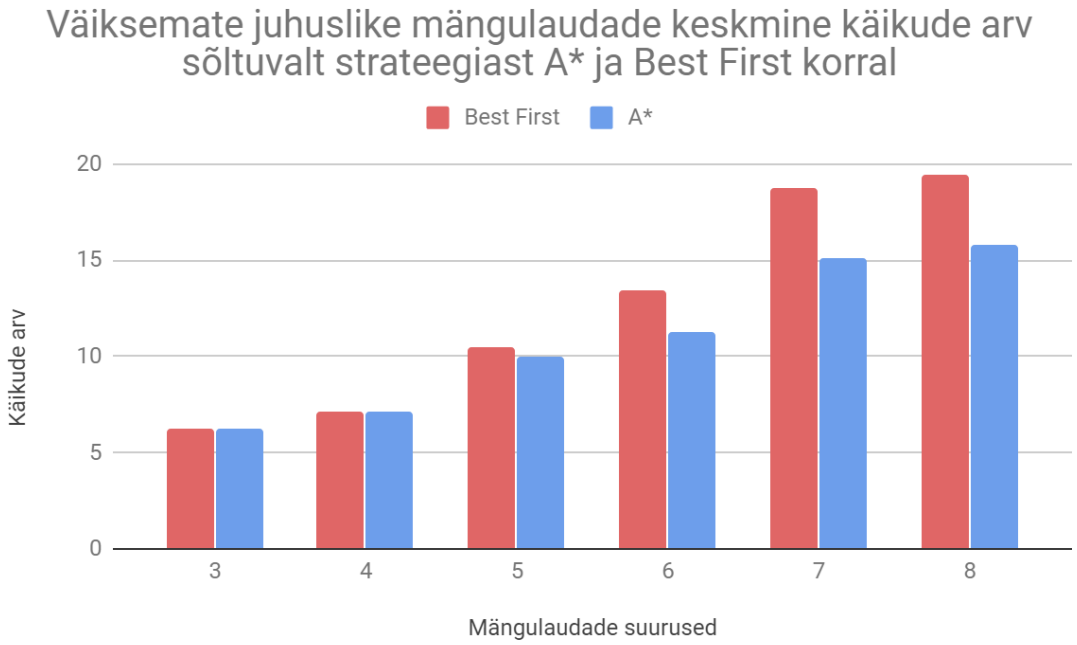
Mõlemat strateegiat käivitati iga lauasuuruse kohta 20 korda, märgiti keskmised lauasuuruse kohta lahenduse leidmise ajad ning saadi järgnevad tulemused, mida illustreerib Joonis 10.



Joonis 10. A* ja Best First strateegiatega väiksemate mängulaudade lahenduvuse võrdlus aja suhtes

Joonis 10 on toodud strateegiate ajalised tulemused. Mõlemad strateegiad jõudsid lahenduseni. On näha, et samade laudade peal leiab *Best First* strateegia lahenduse kordades kiiremini, kui A*. *Best First* puhul jäid kõikide laudade keskmised ajad alla 0,1 sekundi, A* puhul oli aeg mõnevõrra suurem.

Sarnane analüüs tehti ka käikude arvu kohta. Tulemused on toodud Joonis 11.



Joonis 11. A* ja *Best First* strateegiatega väiksemate mängulaudade lahenduvuse võrdlus käikude suhtes

Selgus, et kui väiksemate mängulaudade puhul oli erinevate strateegiatega leitud lahenduskäik sama pikkusega, siis suuremate laudade puhul leidis *Best First* pikema lahenduse, kui A*. Kuna *Best First* strateegia ei võta arvesse käikude arvu, siis ei leia see ka optimaalset lahendust. A* leidis käikude arvu poolest sama optimaalse lahenduse kui *Brute Force*.

Võrreldes strateegiate poolt lahenduskäiguni jõudmise aega, leidis *Best First* strateegia lahenduse kõige kiiremini. Käikude poolest optimaalne A* strateegia leidis lahenduse pikema aja vältel.

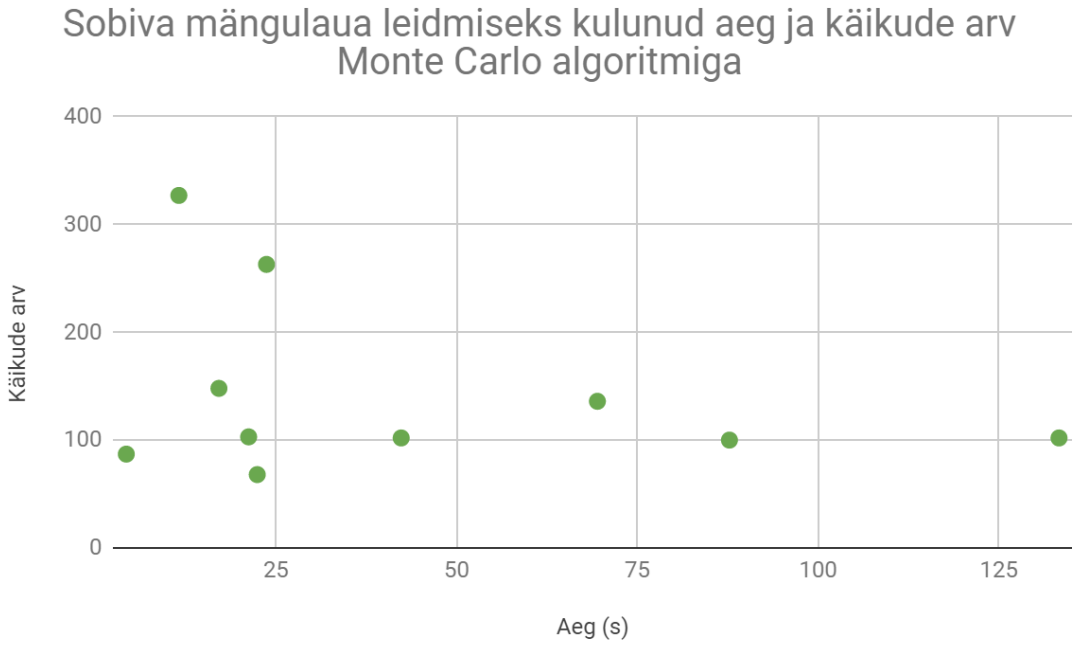
3.3 Originaallaua lahendused

Antud töö algne eesmärk oli analüüsida ka originaallaua lahenduvust läbi erinevate lahendusstrateegiate. Seega tehti kindlaks, kas originaallaud on valitud strateegiate korral lahenduv ning kui jah, siis kuidas.

Brute Force lahendamise võimekust kontrolliti juba käesoleva töö alguses, seega on järgnevad katsed läbi viidud vaid MCTS, A* ja *Best First* strateegiate peal.

3.3.1 *Monte Carlo Tree Search* originaallaual

Originaallaua lahendamiseks jäid kõik varasemalt seatud piirangud samaks. Kuna MCTS puhul on tegemist tõenäosusliku otsinguga, simuleeriti sama alglauaga 10 mängu ning saadi tulemused, mida illustreerib Joonis 12.



Joonis 12. Monte Carlo lahenduse ajad ja käigud originaallaual

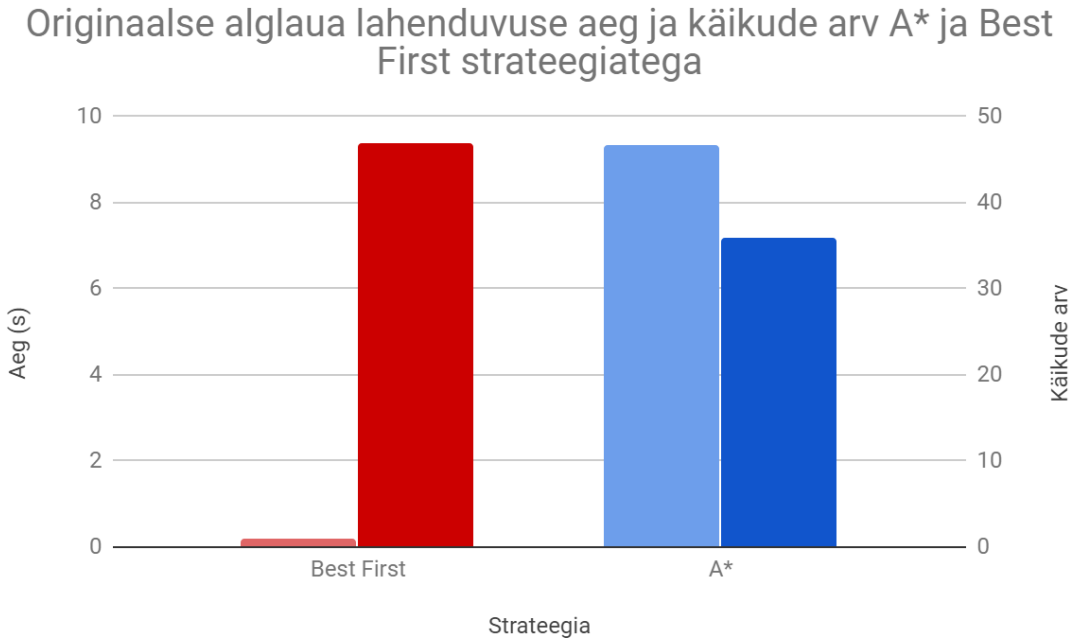
Joonis 12 on näha, et implementeeritud Monte Carlo algoritm suutis originaallauale lahenduse leida keskmiselt üsna sarnase käikude arvuga ning üsna sarnase ajaga. Kümnest katsest kuuel korral suutis strateegia leida lahenduse 100-150 käiguga, kahel korral alla 100 ning kahel korral üle 250 käiguga. Kümnest korrast kuue lahenduse leidmise aeg jäi alla 25 sekundi, ülejäänud neljal oli lahenduse aeg pikem, ulatudes kuni 133 sekundini.

Keskmiselt leiti lahendus 43 sekundi ning 143 käiguga.

3.3.2 A* Search ja *Best First* originaallaual

Originaallaua puhul jäid kõik muud varasemalt kasutatud parameetrid samaks, muutus ainult mängu alglaud ja sellega seotud reeglid. A* ja *Best First* algoritmide eripäradest lähtuvalt on iga sama alglauaga alustatud mäng sama lahenduskäiguga, kui heuristika ei

muutu. Seega viidi antud töös A* ja *Best First* strateegiatega läbi kaks katset – üks katse mõlema strateegia kohta. Tulemused on näha Joonis 13.



Joonis 13. A* ja *Best First* strateegiate lahenduse ajad ja käigud originaallaual

Joonis 13 on näha, et kuigi *Best First* jõudis lahenduseni suurema arvu käikudega (47 käiku) kui A* (36 käiku), oli selle strateegia lahenduseni jõudmise aeg kordades väiksem (0,1 sek) kui A* (9,3 sek).

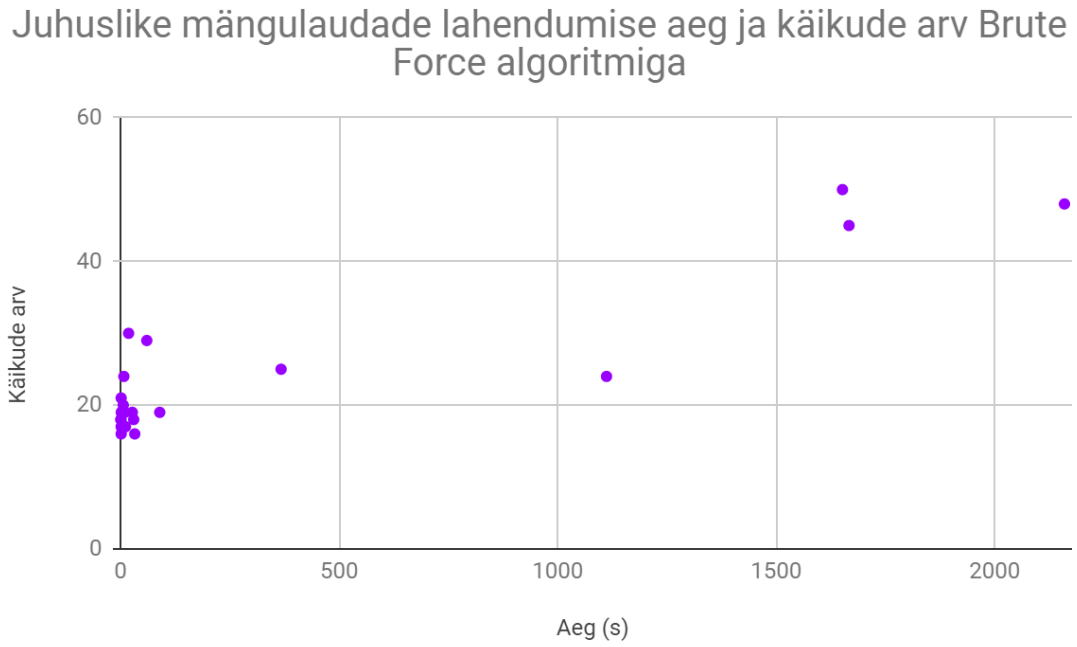
3.4 Lahendused suurematel juhuslikel laudadel

Selleks, et veelgi paremini hinnata ja üldistada erinevate algoritmide oskust võimalikult kiiresti võidulauani jõuda, viidi läbi katsed ka juhuslikult genereeritud laudadega. Ainus jääv reegel olid mängulaua suurus, mis oli alati alguses 27 numbrit ning reegel, kus sarnaselt originaalmängule saab eemaldada paare vaid siis, kui paarilised on samaväärsed või annavad summaks kokku 10. Muus osas olid mängulauad ning nendes sisalduvad numbrid ning nende järjekord juhuslikud.

Katsed viidi läbi *Brute Force*, MCTS, A* ning *Best First* algoritmide strateegiatega 30 korral samade alglaudadega. Töös kasutatud mängulauad on toodud Tabel 1, Lisa 1.

3.4.1 *Brute Force* juhuslikul mängulaual

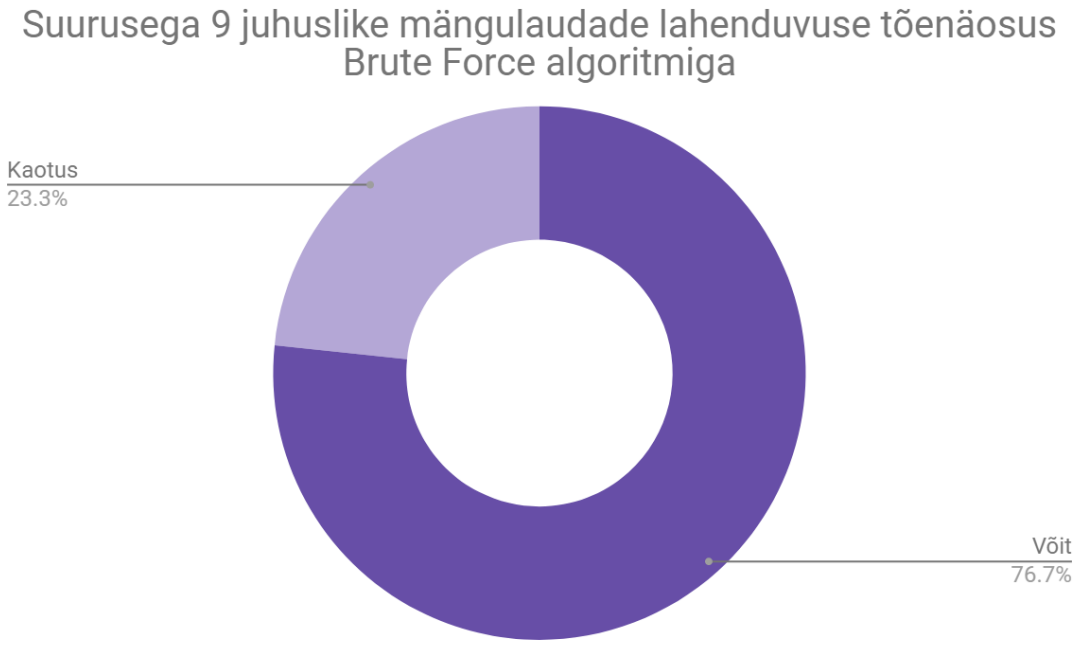
Rakendades *Brute Force* algoritmi 30 suvalisel laual suurusega 9, saadi tulemused, mis on toodud Tabel 2, Lisa 1. Tabeli põhjal loodi Joonis 14.



Joonis 14. Juhuslike mängulaudade lahendumise aeg ja käikude arv *Brute Force* algoritmiga

Jooniselt on näha, et juhuslike mängulaudade puhul leiab *Brute Force* algoritm soovitud tulemuse üldiselt vähem kui minuti ja umbes 20 käiguga. Samas leidub mängulaudu, mille lahendamise aeg võib ulatuda 35 minutini, kuigi tehtud käikude arv nendes mängudes keskmisest väga palju kõrgem ei olnud.

Joonisel on kujutatud vaid lahenduse leidnud mängude tulemusi. Tabelist on näha, et *Brute Force* algoritm leidis lahenduse juhuslikule mängulauale suurusega 9 kokku 23 korral. Viiel korral muutus mälukasutus liialt suureks ning lahendit ei leitud, kahel korral jõuti 100 käigu piirini (Tabel 1, Lisa 1, mängulauad 15 ja 17) ning seetõttu samuti lahendit ei leitud. Seda illustreerib Joonis 15.



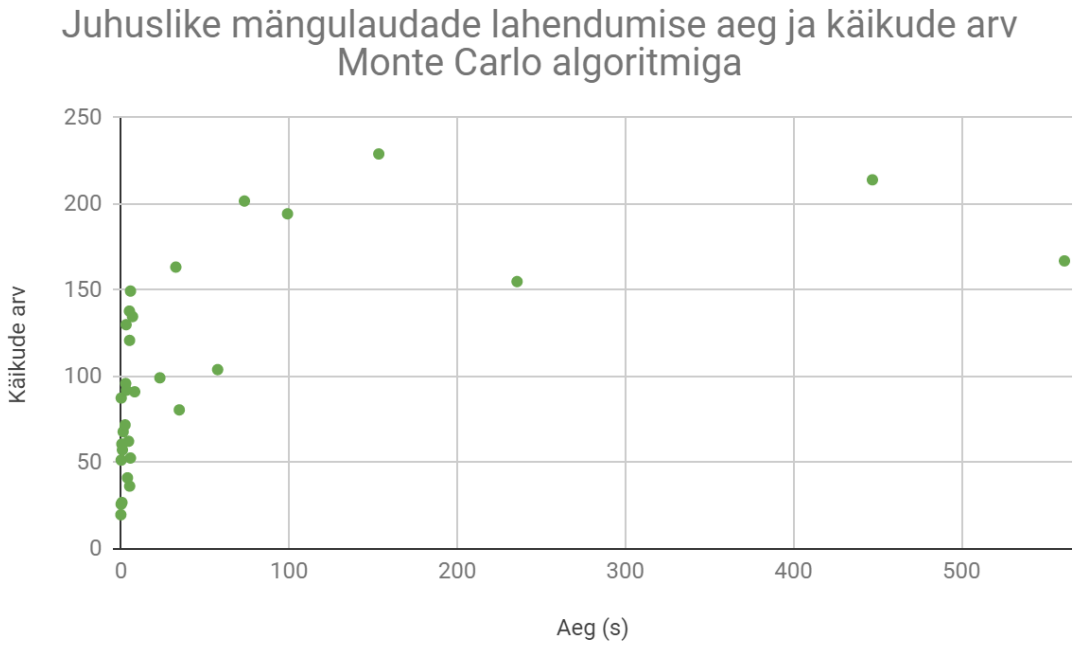
Joonis 15. Juhuslike mängulaudade lahenduvuse tõenäosus *Brute Force* algoritmiga

Keskmine aeg, mille jooksul *Brute Force* soovitud lahenduse leidis oli 315,7 sekundit ning keskmine käikude arv 23,8.

3.4.2 *Monte Carlo Tree Search* juhuslikul mängulaual

MCTS'ga viidi samuti läbi 30 katset, iga mängulauaga 10 korda, mille koondtulemusi kirjeldab Tabel 3, Lisa 1.

Saadud tabeli põhjal loodi Joonis 16, millel on kujutatud mängulaudade lahenduste leidmiseks kulunud keskmised ajad ja keskmised käikude arvud.



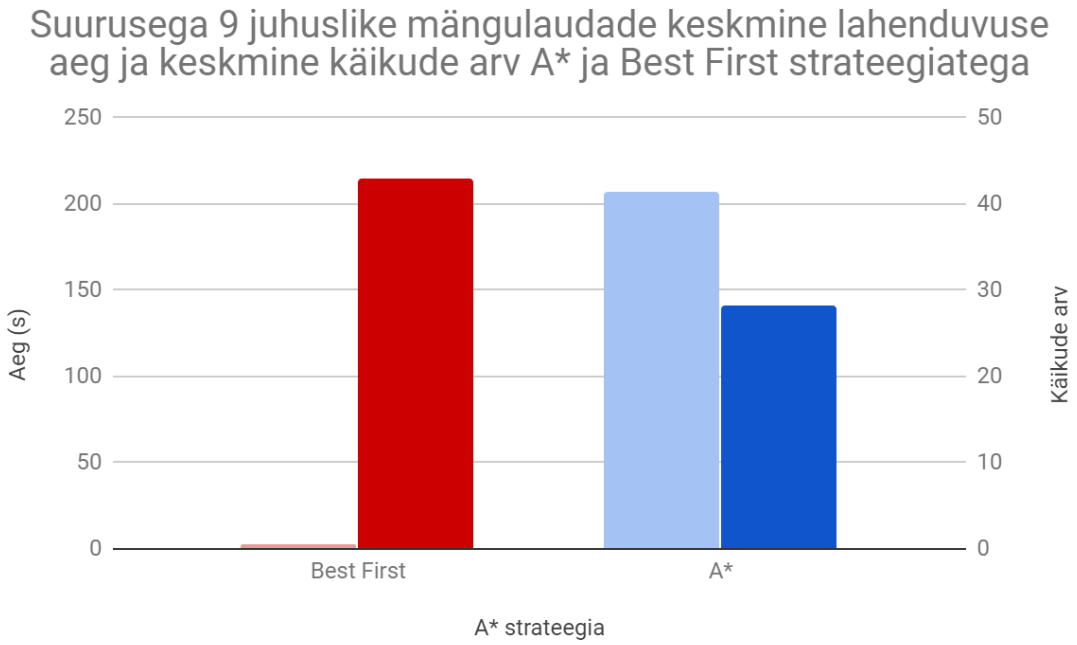
Joonis 16. Juhuslike mängulaudade lahendumise keskmine aeg ja käikude arv Monte Carlo algoritmiga

Jooniselt on näha, et algoritm lahendas mängulaua üldiselt ära vähem kui 100 käiguga ning vähem kui 15 sekundiga. Siiski leidus mänge, mis osutusid keerulisemateks. Sellised mängud lahendas algoritm rohkem kui 100 sekundiga. Maksimaalne lahendusaeg oli 561 sekundit (mängulaud 17, Tabel 1, Lisa 1), kuid käikude arv sel mängul vaid 167. Kõige rohkem käike, 229, tehti mängulaual number 15 (Tabel 1, Lisa 1) 153 sekundiga.

Nende kahe katse pealt on näha, et lahendusaeg ja käikude arv on sõltuvad tõenäosuslikust otsingust – kui esimesel sammul antakse järgnevatele sammudele õige hinnang, jõutakse optimaalsema aja ning käikude arvuga lõpplahenduseni. Samuti võib eeldada, et kuna mõlema mängulaua keskmised tulemused on suhteliselt suured, on tegu keerulisemate laudadega.

3.4.3 A* Search ja *Best First* juhuslikul mängulaual

Selleks, et oleks võimalik võrrelda omavahel erinevate strateegiate lahendamist, viidi A* ja *Best First* strateegiate katsed läbi samade laudadega, mis *Brute Force* ja MCTS puhul kasutusel olid. Saadud tulemused on kujutatud Tabel 4 ja Tabel 5, Lisa 1 vastavalt strateegiale ja kokkuvõtvalt Tabel 8, Lisa 1.



Joonis 17. Juhuslike mängulaudade keskmine lahenduvuse aeg ja keskmine käikude arv A* ja *Best First* strateegiatega

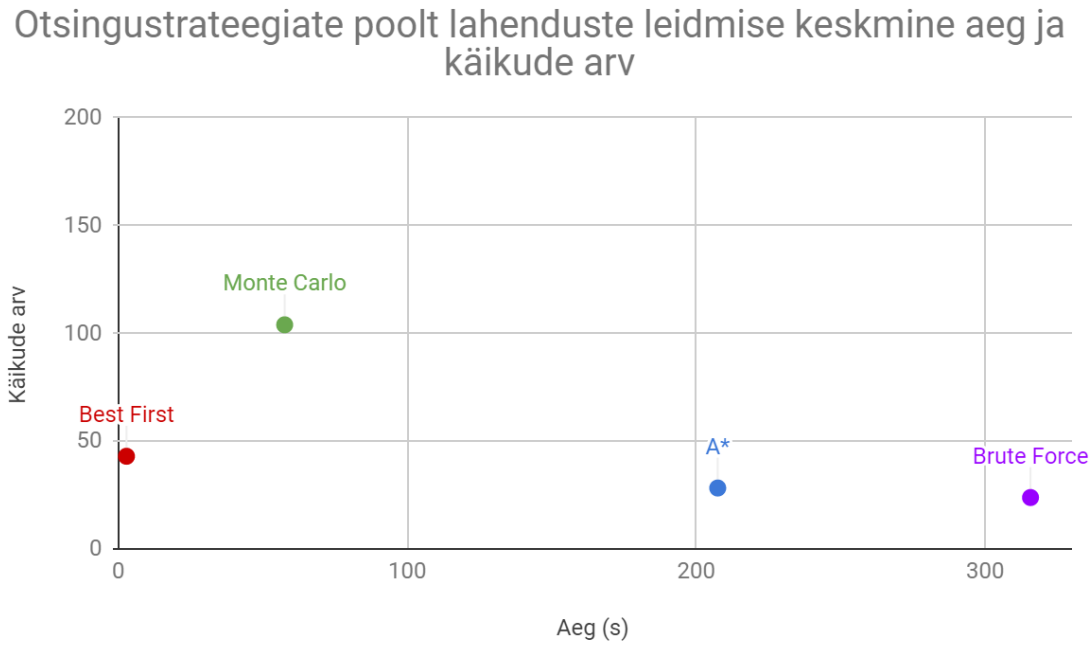
Jooniselt on näha, et *Best First* strateegia on keskmiselt kõige optimaalsema lahenduvuse ajaga (~2,8 sekundit), kuid seejuures jõuab teha kümneid käike rohkem kui A* strateegia. Keskmiselt kõige pikemalt otsis lahendust A* algoritm (~207,4 sekundit), kuid tegi seejuures keskmiselt vaid 28 käiku.

Lisaks selgus, et *Best First* ja A* strateegiad leidsid lahenduse kõigile 30 lauale.

3.5 Võrdlus

Antud töö eesmärgiks oli võrrelda ja analüüsida erinevate algoritmide ning lahendusstrateegiate võimekust ning parameetreid, et leida parim strateegia mängu TenPair reeglitega mängu lahendamiseks, seda nii ajaliselt kui käikude arvu suhtes optimaalselt.

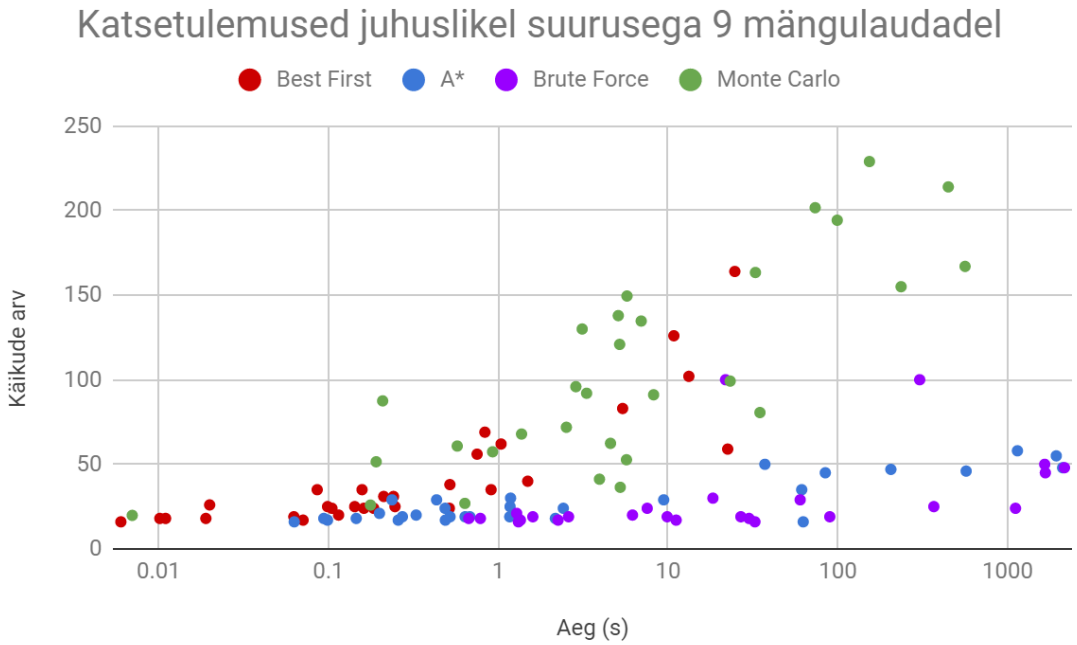
Selle jaoks leiti keskmised ajad ning käikude arvud suurusega 9 juhuslikel laudadel, mille tulemused on toodud Joonis 18. Koondandmed kõikide töös kasutatud strateegiate kohta on toodud Tabel 6 ja Tabel 7, Lisa 1, vastavalt aja ja käikude arvu järgi ning koondtabel mõlema kohta Tabel 8, Lisa 1. Neid tulemusi kirjeldab Joonis 19.



Joonis 18. Otsingustrateegiate poolt lahenduste leidmise keskmine aeg ja käikude arv

Joonis 18 võib näha erinevate otsingustrateegiate lahenduste leidmise keskmiste aegade ning keskmiste käikude arvu võrdlust. Võib näha, et ajaliselt kõige kiiremini jõudis lahenduseni *Best First* strateegia. Keskmiselt kõige kauem otsis lahendust *Brute Force*, mis oli ka eeldatav. Käikude arvu suhtes on optimaalseimad A* ning *Brute Force* algoritmid. Kõige rohkem käike lahenduse leidmiseks tegi Monte Carlo algoritm.

Joonisel on toodud vaid lahenduseni jõudnud tulemused. *Brute Force* jõudis tulemuseni 23 korral.



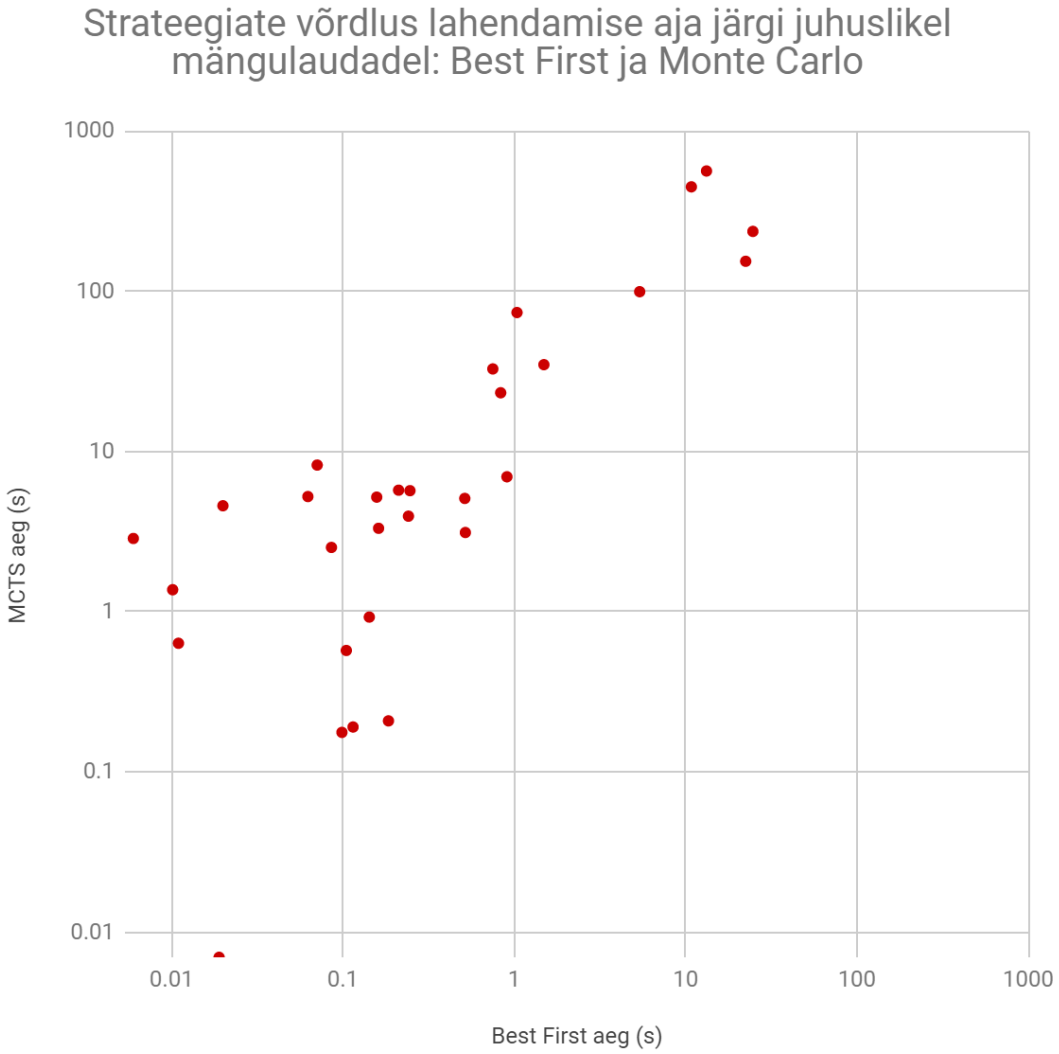
Joonis 19. Otsingustrateegiate poolt lahenduste leidmise aeg ja käikude arv

Joonis 19 näeb kõigi nelja strateegiaga kõiki lahenduseni viinud mängude tulemusi. Kokkuvõtvalt võib öelda, et kuigi kõik strateegiad on üksteisest erinevad, on nende tulemustes siiski sarnaseid lahendusaegu ning käikude arve. Strateegiate tulemused on üsna hajusad ning kindlat parimat ei paista.

Mänguseisudest, millele kõik strateegiad suutsid lahenduse leida, leiti sama käikude arvuga tulemus A* ja *Brute Force* puhul 23 korral. Ülejäänud seitsmel korral ei jõudnud *Brute Force* lahenduseni. *Best First* leidis sama käikude arvuga lahenduse kuuel korral.

Võrdlemaks strateegiaid omavahel veelgi, koostati joonised nii kahe algoritmi lahendamisaja kui kahe algoritmi käikude hulga põhjal. Joonis 18 loetud andmete põhjal tehti ajaline võrdlus *Best First* ja MCTS strateegiate peal, käikude võrdlus *Best First* ja A* strateegiate peal.

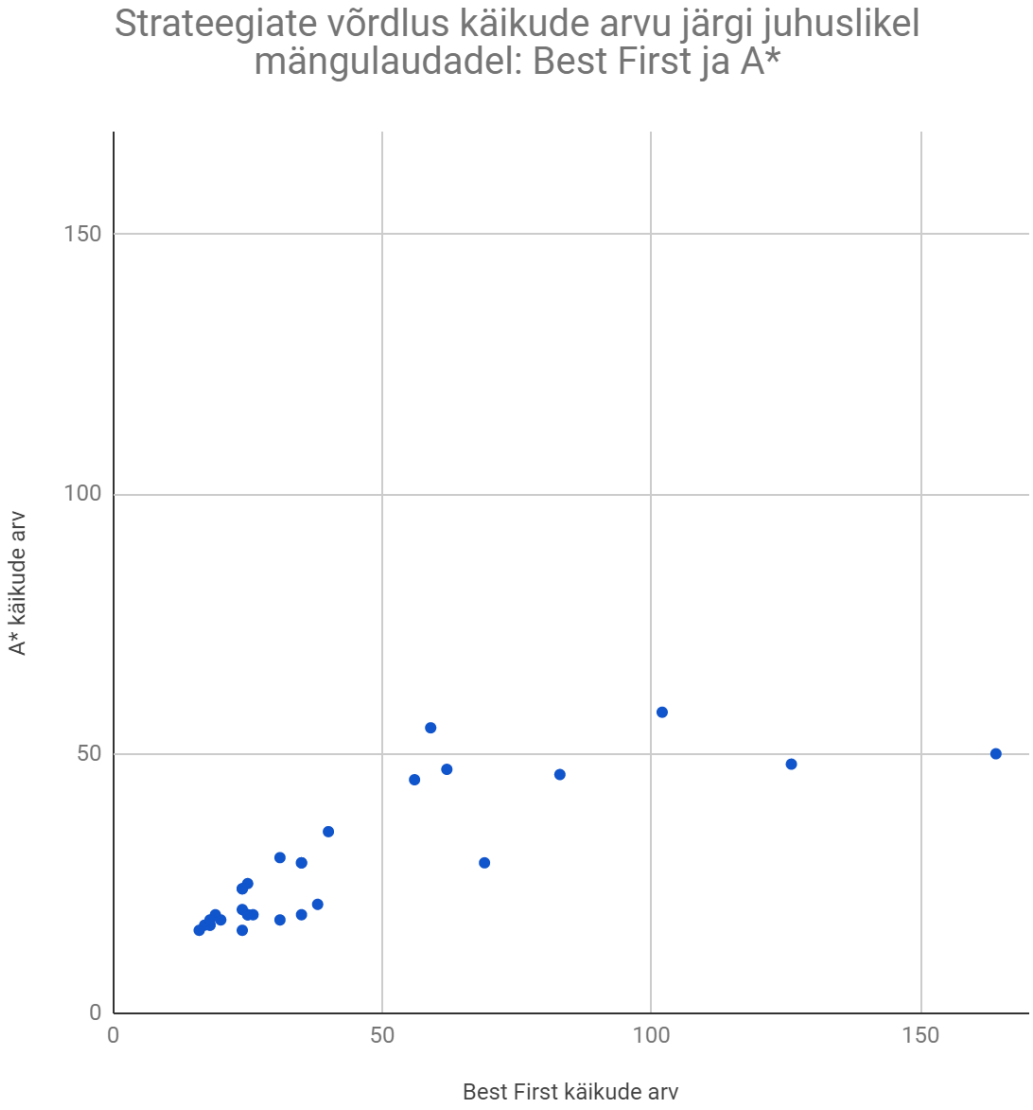
Best First ja MCTS ajalist võrdlust esitab Joonis 20.



Joonis 20. Juhuslike mängulaudade lahendamise võrdlus aja järgi *Best First* ja MCTS strateegiatel

Joonis 20 võib näha, et *Best First* strateegia on aja suhtes optimaalsem kui MCTS strateegia. *Best First* puhul leiti lahendus vähem kui 1 sekundiga 23 korral, MCTS seevastu vaid 7 korral. *Best First* strateegia maksimaalne laua lahendamise aeg oli 235 sekundit, MCTS pikim lahendusaeg oli peaaegu poole pikem – 561 sekundit. Leidus üks mängulaud, mille mõlemad strateegiad lahendasid ära vähem kui 0,02 sekundiga (mängulaud 25, Tabel 1, Lisa 1).

Käikude hulga *Best First* ja A* vahelist võrdlust esitab Joonis 21. Valitud sai A* strateegia *Brute Force* strateega asemel, kuna viimane ei lahendanud lõpuni kõiki laudu. A* seda tegi.



Joonis 21. Juhuslike mängulaudade lahendamise võrdlus käikude järgi *Best First* ja A* strateegiatel

Selgus, et 30 mängust kuuel korral olid strateegiate teekonnad võrdse pikkusega. Ülejäänud kordadel oli A* strateegia käikude arv alati madalam, kui *Best First* strateegia oma. Maksimaalne käikude arv *Best First* strateegia puhul oli 164, A* puhul 58. Vähem kui 50 käiguga suudeti lahendada 21 mängulauda.

Mängu TenPair saab seega lahendada üsna mitmete algoritmide ja strateegiatega. Kõikidel algoritmidel ja strateegiatel on omad eripärad. Ajaliselt kõige optimaalsema lahenduse nii originaallauale kui suvalisele alglauale leiab *Best First* strateegia. Käikude hulka analüüsides on optimaalsem A* strateegia. Kõige kauem otsib võiduseisu *Brute*

Force algoritm, kuna otsingupuu on lai ning kõikide võimalike järglaste läbi käimine võib võtta pikalt aega. Keskmiselt kõige rohkem käike teeb MCTS strateegia, kuid kuna tegu on tõenäosusliku otsinguga, võib lahenduse leida hea õnne korral ka väiksema käikude arvu ja üsnagi optimaalse ajaga.

4 Kokkuvõte

Matemaatilisi mänge on mitmeid ja nii mõnegi kohta on tehtud uuringuid ja nende jaoks loodud lahendusstrateegiaid. Mäng TenPair on üsnagi vähetuntud ning seetõttu pole seda ka põhjalikumalt uuritud.

Käesoleva töö eesmärk oli analüüsida mängu TenPair lahendamiseks sobivaid strateegiaid, hinnata nende võimekust ning nende kaudu mängu lahenduvust. Töös tutvuti põhjalikumalt nelja algoritmi – *Brute Force*, *Monte Carlo Tree Search*, A*, *Best First* - strateegiatega ning prooviti neid mängu lahendamisel. Katsed sooritati nii väiksematel laudadel, originaal-alglaual kui juhuslikult genereeritud suurematel laudadel. Tulemusi analüüsiti peamiselt ajakasutuse ning lahenduseks kulunud käikude hulga põhjal.

Selgus, et väiksemaid mängulaudasid lahendavad kõik valitud strateegiad üsnagi optimaalse aja ja käikude arvuga. Seevastu originaallaua suutsid lahendada vaid *Best First*, A* ja Monte Carlo strateegiad. Algoritmide tööd hinnati ka suurematel mängulaudadel, mis olid genereeritud juhuslikult ning neid mängulaudu lahendas kõige aja suhtes kõige optimaalsemalt *Best First* strateegia.

Seega võib antud töö põhjal arvata, et parim lahendusstrateegia vaadeldud strateegiatest mängu „TenPair“ lahendamiseks on *Best First*, kuna vaatamata lahenduste pikkusele leiti need lahendused kõige lühema ajaga.

A* suudab leida optimaalse lahenduse käikude arvu mõttes kõigile vaadeldud laudadele, mis tähendab, et mäng on sellega läbi arvutatud ning optimaalsed strateegiad võib igale mängulauale ette arvutada.

Ükski strateegia polnud siiski ideaalne ning piisavalt detailne, et sobida antud mängu lahendamiseks igas olukorras. Vaadeldud algoritmid olid üsnagi lihtsad ja üldtuntud, kuid kuna antud mängu polnud varem üldse uuritud, tundus esimeseks katsetuseks sellest piisavat.

Antud teemaga saaks minna veel rohkem süvitsi ning uurida detailsemalt ka teiste algoritmide ja strateegiate lahendamise võimekust ning parameetreid, näiteks mõningaid MinMax või Alfa-Beeta otsingu strateegiaid. Samuti leida ainulaadsemaid ja universaalsemaid lahendusi ka käesolevas töös välja toodud meetoditele.

Kasutatud kirjandus

[1] “Heuristika,” [Online]. Available: https://et.wikipedia.org/wiki/Heuristika. [Accessed 24 Apr 2019].

[2] “Memoization,” [Online]. Available: https://en.wikipedia.org/wiki/Memoization. [Accessed 06 May 2019].

[3] “Stiimulõpe,” [Online]. Available: https://et.wikipedia.org/wiki/Stiimul%C3%B5pe. [Accessed 20 May 2019].

[4] “Brute-force Search,” [Online]. Available: https://en.wikipedia.org/wiki/Brute-force_search. [Accessed 06 April 2019].

[5] S. Sharma, “Monte Carlo Tree Search,” [Online]. Available: https://towardsdatascience.com/monte-carlo-tree-search-158a917a8baa. [Accessed 6 April 2019].

[6] A. Choudhary, “Introduction to Monte Carlo Tree Search: The Game-Changing Algorithm behind DeepMind’s AlphaGo,” [Online]. Available: https://www.analyticsvidhya.com/blog/2019/01/monte-carlo-tree-search-introduction-algorithm-deepmind-alphago/. [Accessed 6 April 2019].

[7] M. Campbell, “Mastering board games,” *Science,* vol. 362, no. 6419, 2018.

[8] R. Rahul, “ML | Monte Carlo Tree Search (MCTS),” [Online]. Available: https://www.geeksforgeeks.org/ml-monte-carlo-tree-search-mcts/. [Accessed 7 April 2019].

[9] P. Sinclair, “MCTS package,” [Online]. Available: https://pypi.org/project/mcts/. [Accessed 14 April 2019].

[10] “A* search algorithm,” [Online]. Available: https://en.wikipedia.org/wiki/A*_search_algorithm. [Accessed 15 April 2019].

[11] N. Swift, “Easy A* (star) Pathfinding,” [Online]. Available: https://medium.com/@nicholas.w.swift/easy-a-star-pathfinding-7e6689c7f7b2. [Accessed 15 April 2019].

[12] J. Rialland. [Online]. Available: https://pypi.org/project/astar/. [Accessed 20 April 2019].

Lisa 1 – Strateegiate tööd kirjeldavad tabelid

Tabel 1. Juhuslikud mängulauad suurusega 9

Mängulaua number	Mängulaua algseis
1	22253353413443755394581258 4
2	42785891731736477389912179 5
3	31845437999175616125286117 6
4	76233934266824341669797471 4
5	75917774396968651386116645 9
6	51517224489638127987885518 5
7	72951739488135268173288363 8
8	32363364373633431334468161 2
9	15426515917968975522941492 4
10	68768225278122819454221859 2
11	49929845749869812249849653 3
12	45624459964689239761355757 5
13	89962189216792373274634932 1
14	29397695613284858749486434 6
15	62765526297629212894947471 5
16	61465211343663173957589797 5
17	42935388581638683767225746 5
18	33573276238651188994379455 4
19	16898214289546613722379418 2
20	96964539995342711714995797 3
21	95622728295985572233489627 8
22	81359194746282717629766687 8
23	11831692718786281322692614 8
24	12441881735316616221161793 4
25	98452822754167373195934929 2
26	41318841344721619878997813 3
27	99226966894664984491464424 6
28	94345871416264574188967553 7
29	74718136667112814512833743 7
30	73796938389548197581465157 7

Tabel 2. Juhuslike suurusega 9 mängulaudade lahenduvus *Brute Force* algoritmiga

Mängulaua number	Lahenduse leidmiseks kulunud aeg (s)	Käikude arv	Võit?
1	2.591765165	19	TRUE
2	1.348747969	17	TRUE
3	2.247641802	17	TRUE
4	11.15190864	17	TRUE
5	inf	inf	FALSE
6	59.94420385	29	TRUE
7	inf	inf	FALSE
8	29.96044564	18	TRUE
9	inf	inf	FALSE
10	1651.681781	50	TRUE
11	1111.807654	24	TRUE
12	1.596729517	19	TRUE
13	0.6692094803	18	TRUE
14	1666.656524	45	TRUE
15	303.2523854	100	FALSE
16	9.876586437	19	TRUE
17	21.80069256	100	FALSE
18	2159.561166	48	TRUE
19	367.2474697	25	TRUE
20	inf	inf	FALSE
21	26.82325935	19	TRUE
22	7.53285408	24	TRUE
23	1.313151598	16	TRUE
24	18.36189961	30	TRUE
25	0.7839021683	18	TRUE
26	1.282570124	21	TRUE
27	32.45619226	16	TRUE
28	89.60135269	19	TRUE
29	inf	inf	FALSE
30	6.169499397	20	TRUE

Tabel 3. Juhuslike suurusega 9 mängulaudade lahenduvus Monte Carlo algoritmiga

Mängulaua number	Lahenduse leidmiseks kulunud aeg (s)	Käikude arv	Võit?
1	5.226848197	36.4	TRUE
2	0.6352695227	26.9	TRUE
3	8.214270806	91.1	TRUE
4	1.370555925	67.9	TRUE
5	5.183064699	120.9	TRUE
6	23.20454805	99.2	TRUE
7	73.48295057	201.7	TRUE
8	0.1908133745	51.5	TRUE
9	99.11327033	194.3	TRUE
10	235.6277902	155	TRUE
11	5.088260269	137.9	TRUE
12	0.1765923977	25.9	TRUE
13	3.942895079	41.2	TRUE
14	32.68864691	163.4	TRUE
15	153.3858991	229	TRUE
16	2.516953468	71.9	TRUE
17	561.1655571	167	TRUE
18	446.9598899	214	TRUE
19	5.686313057	52.7	TRUE
20	34.74246051	80.6	TRUE
21	4.579348779	62.4	TRUE
22	0.5729866743	60.8	TRUE
23	2.861878133	95.9	TRUE
24	5.72624824	149.5	TRUE
25	0.006981277466	19.8	TRUE
26	3.11818192	130	TRUE
27	0.2083611488	87.5	TRUE
28	0.9252701283	57.4	TRUE
29	6.942532921	134.7	TRUE
30	3.312014174	92	TRUE

Tabel 4. Juhuslike suurusega 9 mängulaudade lahenduvus *Best First* strateegiaga

Mängulaua number	Lahenduse leidmiseks kulunud aeg (s)	Käikude arv	Võit?
1	0.06248688698	19	TRUE
2	0.0109744072	18	TRUE
3	0.07081007957	17	TRUE
4	0.0101518631	18	TRUE
5	0.1575789452	35	TRUE
6	0.8335838318	69	TRUE
7	1.038089991	62	TRUE
8	0.1146392822	20	TRUE
9	5.398067474	83	TRUE
10	24.71489739	164	TRUE
11	0.5146226883	24	TRUE
12	0.09873747826	25	TRUE
13	0.2413544655	31	TRUE
14	0.7499930859	56	TRUE
15	22.43998432	59	TRUE
16	0.08576893806	35	TRUE
17	13.25155783	102	TRUE
18	10.81108332	126	TRUE
19	0.2463407516	25	TRUE
20	1.490018129	40	TRUE
21	0.01994657516	26	TRUE
22	0.1047165394	24	TRUE
23	0.005984067917	16	TRUE
24	0.2114355564	31	TRUE
25	0.01894950867	18	TRUE
26	0.5186114311	38	TRUE
27	0.1845054626	24	TRUE
28	0.142619133	25	TRUE
29	0.9065754414	35	TRUE
30	0.1615681648	24	TRUE

Tabel 5. Juhuslike suurusega 9 mängulaudade lahenduvus A* strateegiaga

Mängulaua number	Lahenduse leidmiseks kulunud aeg (s)	Käikude arv	Võit?
1	0.5196094513	19	TRUE
2	0.4876499176	17	TRUE
3	0.2573108673	17	TRUE
4	0.09865188599	17	TRUE
5	0.4329822063	29	TRUE
6	0.2364304066	29	TRUE
7	205.0906765	47	TRUE
8	2.168010473	18	TRUE
9	569.4499607	46	TRUE
10	37.15238237	50	TRUE
11	2.414540529	24	TRUE
12	0.6372959614	19	TRUE
13	0.1456103325	18	TRUE
14	84.317693	45	TRUE
15	1933.06167	55	TRUE
16	0.2722756863	19	TRUE
17	1141.507339	58	TRUE
18	2106.160702	48	TRUE
19	1.172861576	25	TRUE
20	61.22823811	35	TRUE
21	1.167878151	19	TRUE
22	0.4857003689	24	TRUE
23	0.06283092499	16	TRUE
24	1.180839777	30	TRUE
25	0.09374928474	18	TRUE
26	0.1994659901	21	TRUE
27	62.48487735	16	TRUE
28	0.6801805496	19	TRUE
29	9.419805288	29	TRUE
30	0.3281223774	20	TRUE

Tabel 6. Strateegiate koondtabel mängulaudadel suurusega 9 aja järgi

Mängulaua number	*Best First*	A*	*Brute Force*	Monte Carlo
1	0.06248	0.51960	2.59176	5.22684
2	0.010974	0.487649	1.34874	0.63526
3	0.070810	0.257310	2.24764	8.21427
4	0.010151	0.098651	11.15190	1.37055
5	0.157578	0.432982	inf	5.18306
6	0.8335838	0.236430	59.94420	23.20454
7	1.03808	205.09067	inf	73.48295
8	0.11463	2.16801	29.96044	0.19081
9	5.39806	569.44996	inf	99.11327
10	24.71489	37.15238	1651.68178	235.62779
11	0.514622	2.41454	1111.80765	5.08826
12	0.09873	0.63729	1.59672	0.17659
13	0.24135	0.14561	0.66920	3.94289
14	0.74999	84.31769	1666.65652	32.68864
15	22.43998	1933.06167	303.25238	153.38589
16	0.085768	0.272275	9.876586	2.516953
17	13.25155	1141.50733	21.80069	561.16555
18	10.81108	2106.16070	2159.56116	446.95988
19	0.246340	1.172861	367.24746	5.68631
20	1.49001	61.22823	inf	34.74246
21	0.019946	1.167878	26.82325	4.57934
22	0.104716	0.48570	7.53285	0.57298
23	0.00598	0.06283	1.31315	2.86187
24	0.211435	1.18083	18.36189	5.72624
25	0.018949	0.093749	0.783902	0.0069812
26	0.518611	0.19946	1.28257	3.11818
27	0.184505	62.48487	32.45619	0.20836
28	0.14261	0.68018	89.60135	0.92527
29	0.90657	9.41980	inf	6.94253
30	0.16156	0.32812	6.16949	3.31201
Keskmine	2.82052	207.43051	315.68115	36.15375

Tabel 7. Strateegiate koondtabel mängulaudadel suurusega 9 käikude hulga järgi

Mängulaua number	*Best First*	A*	*Brute Force*	Monte Carlo
1	19	19	19	36.4
2	18	17	17	26.9
3	17	17	17	91.1
4	18	17	17	67.9
5	35	29	inf	120.9
6	69	29	29	99.2
7	62	47	inf	201.7
8	20	18	18	51.5
9	83	46	inf	194.3
10	164	50	50	155
11	24	24	24	137.9
12	25	19	19	25.9
13	31	18	18	41.2
14	56	45	45	163.4
15	59	55	100	229
16	35	19	19	71.9
17	102	58	100	167
18	126	48	48	214
19	25	25	25	52.7
20	40	35	inf	80.6
21	26	19	19	62.4
22	24	24	24	60.8
23	16	16	16	95.9
24	31	30	30	149.5
25	18	18	18	19.8
26	38	21	21	130
27	24	16	16	87.5
28	25	19	19	57.4
29	35	29	inf	134.7
30	24	20	20	92
Keskmine	42.96	28.23	23.83	97.23

Tabel 8. Strateegiate koondtabel mängulaudadel suurusega 9

Mängulaua number	*Best First* aeg (s)	*Best First* käigud	A*1 aeg (s)	A*1 käigud	*Brute Force* aeg (s)	*Brute Force* käigud	Monte Carlo aeg (s)	Monte Carlo käigud
1	0.06248	19	0.51960	19	2.59176	19	5.22684	36.4
2	0.010974	18	0.487649	17	1.34874	17	0.63526	26.9
3	0.070810	17	0.257310	17	2.24764	17	8.21427	91.1
4	0.010151	18	0.098651	17	11.15190	17	1.37055	67.9
5	0.157578	35	0.432982	29	inf	inf	5.18306	120.9
6	0.8335838	69	0.236430	29	59.94420	29	23.20454	99.2
7	1.03808	62	205.0906	47	inf	inf	73.48295	201.7
8	0.11463	20	2.16801	18	29.96044	18	0.19081	51.5
9	5.39806	83	569.4499	46	inf	inf	99.11327	194.3
10	24.71489	164	37.15238	50	1651.68178	50	235.62779	155
11	0.514622	24	2.41454	24	1111.80765	24	5.08826	137.9
12	0.09873	25	0.63729	19	1.59672	19	0.17659	25.9
13	0.24135	31	0.14561	18	0.66920	18	3.94289	41.2
14	0.74999	56	84.31769	45	1666.65652	45	32.68864	163.4
15	22.43998	59	1933.061	55	303.25238	100	153.38589	229
16	0.085768	35	0.272275	19	9.876586	19	2.516953	71.9
17	13.25155	102	1141.507	58	21.80069	100	561.16555	167
18	10.81108	126	2106.160	48	2159.56116	48	446.95988	214
19	0.246340	25	1.172861	25	367.24746	25	5.68631	52.7
20	1.49001	40	61.22823	35	inf	inf	34.74246	80.6
21	0.019946	26	1.167878	19	26.82325	19	4.57934	62.4
22	0.104716	24	0.48570	24	7.53285	24	0.57298	60.8
23	0.00598	16	0.06283	16	1.31315	16	2.86187	95.9
24	0.211435	31	1.18083	30	18.36189	30	5.72624	149.5
25	0.018949	18	0.093749	18	0.783902	18	0.0069812	19.8
26	0.518611	38	0.19946	21	1.28257	21	3.11818	130
27	0.184505	24	62.48487	16	32.45619	16	0.20836	87.5
28	0.14261	25	0.68018	19	89.60135	19	0.92527	57.4
29	0.90657	35	9.41980	29	inf	inf	6.94253	134.7
30	0.16156	24	0.32812	20	6.16949	20	3.31201	92
Keskmine	2.82052	42.96	207.4305	28.23	315.68115	23.83	36.15375	97.23