TARTU ÜLIKOOL
MATEMAATIKA-INFORMAATIKATEADUSKOND
Arvutiteaduse instituut
Informaatika eriala

Mari-Liis Oldja

Mäkketõusu algoritm ja selle variandid kahe probleemi näitel

Bakalaureusetöö (6 EAP)

Juhendaja: prof. Mare Koit

TARTU 2014

Mäkketõusu algoritm ja selle variandid kahe probleemi näitel

Lühikokkuvõte:

Mäkketõusu algoritm on üks lihtsamaid tehisintellekti otsingualgoritme ja seetõttu ka tihti kasutatav. Antud töö eesmärk on Tartu Ülikoolis õpetatava tehisintellekti kursuse kuulajatele lisamaterjali loomine, mis aitaks mäkketõusu algoritmi ning selle erinevate variantide tööpõhimõtet paremini mõista. Lisaks on töö käigus loodud programm, mis 8-mängu ja 8 lipu probleemi alusel visualiseerib algoritmi tööd ning toetab seega veelgi teemast arusaamist.
Töö sisaldab ka ülevaadet loodud programmi abil kogutud statistikast, mis võiks aidata mäkketõusu algoritmi erinevaid variante ka omavahel võrrelda.

Võtmesõnad: probleemilahendus, heuristiline otsing, mäkketõusu algoritm, 8-mäng, 8 lipu probleem

Hill climbing algorithm and its versions on the example of two problems

Abstract:

Hill climbing algorithm is one of the searching algorithms in artificial intelligence. It is often used because of its simplicity. The goal of this thesis is to produce materials that could aid students taking artificial intelligence course to better understand the concept of hill climbing algorithm. In addition, a program was written that visualizes the use of hill climbing algorithm on the example of 8-puzzle and 8 queens problem. The program should further support the understanding of the given topic.
Statistics collected with the program is used to compare the different versions of hill climbing algorithm and bring out the strengths and weaknesses of the algorithms used.

Keywords: problem solving, heuristic search, hill climbing algorithm, 8-puzzle, 8 queens problem

Autorideklaratsioon

Deklareerin, et käesolev bakalaureusetöö on minu iseseisva töö tulemus, on esitatud Tartu Ülikooli Arvutiteaduse instituudi lõpudiplomi taotlemiseks informaatika erialal. Lõputöö alusel ei ole varem eriala lõpudiplomit taotletud.
Autor /Nimi/ .....

(allkiri ja kuupäev)

Sisukord

Sissejuhatus

Mäkketõusu algoritm on oma lihtsuse tõttu üks kasutatavamaid probleemilahenduse algoritme tehisintellekti valdkonnas. Antud töös tutvustatakse mäkketõusu algoritmi ning selle erinevaid variante, lisaks antakse ülevaade kahest probleemist, mille lahendamisel mäkketõusu algoritmi rakendada saaks. Lähemalt saab algoritmidega tutvuda, kasutades töö käigus valminud programmi, mis nende kahe probleemi näitel demonstreerib samm-sammult algoritmi erinevate variantide tööd. Programmi abiga saab koguda ka statistikat, mille abil algoritme omavahel võrrelda.

Esimeses peatükis (mõisted ja algoritmide kirjeldused) tutvustatakse üldiselt otsingumeetodeid ja teemaga seotud mõisteid ning antakse ülevaade mäkketõusu algoritmi erinevatest variantidest.

Teises peatükis (probleemide tutvustus) antakse ülevaade kahest probleemist, 8-mäng ja 8 lipu probleem, mida tehisintellektis probleemilahendusmeetodite tutvustamisel tihti kasutatakse. Lisaks kirjeldatakse mõlema probleemi puhul võimalikke heuristikuid ning nende eeliseid ja puuduseid.

Kolmandas peatükis (programmi ülevaade) tutvustatakse töö käigus valminud, mäkketõusu algoritmi erinevaid variante illustreerivat programmi, antakse ülevaade programmi võimalustest. Lisaks on selgitatud programmi valmimise käigus tehtud valikuid.

Neljandas peatükis (tulemuste analüüs) esitatakse ja analüüsitakse programmi abiga kogutud statistikat, muuhulgas võrreldakse omavahel nii mäkketõusu algoritmi erinevaid variante kui ka probleeme.

Lisadeks on töö käigus valminud programmid: mäkketõusu algoritmi programm, mis visualiseerib mäkketõusu algoritmi ja selle variantide tööd, ning statistika kogumise programm, mille abiga saab koguda statistikat mäkketõusu algoritmi variantide kohta.

1. Mõisted ja algoritmide kirjeldus

See peatükk sisaldab töös ettetulevate mõistete selgitusi ning üldist informatsiooni tehisintellekti valdkonnas kasutatavate otsingumeetodite kohta. Lisaks on kirjeldatud mäkketõusu algoritmi ning selle erinevaid variante ja välja toodud ka nende eeliseid ja puuduseid.

1.1. Taust ja mõisted

Kuna tehisintellekti probleemid on iseenesest keerukad, siis võetakse tehisintellekti meetoditest tihti kasutusele erinevad otsingu tehnikad. Kõik otsingumeetodid kuuluvad üldjuhul ühte kahest kategooriast:

1) Põhjalik ehk **pimeotsing** – on probleemi lahendamine n-ö jõumeetodil, võimaluste proovimisel ei kasutata olemasolevaid teadmisi.[1,2] Pimeotsingu meetodid on näiteks laiutiotsing, süvitsiotsing ja iteratiivne laskumine või laienemine.
2) Informeeritud ehk **heuristiline otsing** – kogemusel ja arvamusel põhinev probleemilahendus, mida kasutatakse otsingu tõhustamiseks. Halvimal juhul võivad lahendid olla halvad, kuid enamasti on heuristiline otsing siiski efektiivsem kui pimeotsing.[1] Heuristilise otsingu meetodid on näiteks A* algoritm, parim enne-otsing ning ka mäkketõusu algoritm ja selle variandid.

Otsingumeetodite kasutamiseks on vajalik, et oleks olemas eeskiri, kuidas mingit probleemi olekut ehk seisu esitada ja mis määraks ka, kuidas seise eristada ning milline on lõppseis. Lisaks on vaja defineerida käigud ehk operaatorid erinevate seisude vahel liikumiseks ning eeskiri, mille järgi neid rakendada.[3]

Lihtsam on olekute ruumi ette kujutada kui otsingugraafi või otsingupuud, kus iga seis on esitatud tipuna ning tipu alluvateks on seisud, mis on vastavast seisust saadavad mingi lubatud operaatori rakendamisel. Otsingu eesmärgiks on tippe mööda liikudes jõuda algtipust lõpptippu, s.t leida lõppseis ehk lahendus. Edasises töös kasutatakse mõistet vahetud järglasseisud või vahetud **järglased** (ühe sammu kaugusel asuvad järglased)**,** tähistamaks seisust kõigi lubatud operaatorite ühekordsel rakendamisel saadavate seisude hulka. Vastavalt kokkuleppele võivad järglasteks olla ka mitme sammu kaugusel olevad seisud, see tähendab seisud, mille leidmiseks kasutatakse mitut operaatorit järjest.

Heuristik on justkui rusikareegel, see on intuitiivne otsuse tegemise meetod, mida rakendatakse puuduliku info korral. Otsingu algoritmides kasutatakse heuristikut eesmärgiga juhtida

otsingut kasulikus suunas. Heuristiline otsing ei garanteeri edu, kuna otsustatakse puuduliku info põhjal, kuid siiski on heuristiline otsing tihti edukas ning vähendab tunduvalt otsingule kuluvat aega. On aga probleeme, mille puhul on heuristilise funktsiooni väärtuse arvutamise hind kõrgem otsingu enda hinnast, otsingu meetodi ning heuristikute valimisel tuleb sellega arvestada.[1]

Huvitav on ka teada, et tehisintellektis võeti heuristikud kasutusele malemängus ja ülesannete lahendamisel, et teha arvutit „targemaks“, psühholoogias aga kasutatakse heuristikuid selleks, et selgitada, miks inimesed targad ei ole.[4]

Selleks, et heuristiku väärtust mingil seisul teada saada, kasutatakse **heuristilist funktsiooni**. Enamasti annab heuristiline funktsioon hinnangu sellele, kui pikk on tee vastavast tipust lõpp-tipuni ehk lahenduseni. See tähendab, heuristiku väärtus väljendab seda, mitu seisu tuleb lõppseisu jõudmiseks läbida.

Hinnangufunktsioon (ingl k *evaluation function*) annab seisule hinnangu vastavalt otsingu-suuna perspektiivikusele ehk teisisõnu aitab hinnangufunktsioon vastata küsimusele, kui suure tõenäosusega viib seis meid lõppseisule lähemale. Hinnangufunktsioon võib ühe osana sisaldada ka heuristilist funktsiooni, sellisel juhul on tegu heuristilise otsinguga, vastasel juhul, kui hinnangufunktsioon heuristilist funktsiooni ei sisalda, on tegu pimeotsinguga.

Sageli on hinnangufunktsioonil kuju

$$f(n) = g(n) + h(n),$$

kus n on seis, $g(n)$ selle seisu kaugus algseisust ning $h(n)$ heuristilise funktsiooni väärtus kohal *n*.[1]

Kuna mäkketõusu algoritmi puhul, kui otsitakse ainult lahendust, mitte lahenduseni jõudmise teed, ei ole seisu kaugus algseisust oluline, siis on edasises hinnangufunktsiooni all mõeldud funktsiooni

$$f(n) = h(n),$$

kus *n* tähistab seisu, mille hinnangut arvutatakse.

1.2. Algoritmid

1.2.1. Mäkketõusu algoritm

Ajalugu ja algoritmist üldiselt

Mäkketõusu algoritm on otsingualgoritm. See on tsükkel, mis liigub igal käigul kõrgema või parema hinnangu suunas, justkui ülesmäge, millest on ka tulnud algoritmi nimi. Mäkketõusu algoritm ei tee kunagi sammu halvema hinnanguga tipu suunas, vaid liigub kuni algoritmi töö

lõpuni ainult ülesmäge. Seda algoritmi nimetatakse tihti ahneks otsingualgoritmiks kuna käik tehakse parema seisu suunas, mõtlemata kaugemale, mida sealt edasi teha.[5] Töö lõpetatakse, kui tipu järglaste[1] hulgas ei ole ühtegi parema hinnanguga tippu, see tähendab, et on jõutud mäetippu. [6]

Mäkketõusu algoritmi töö käigus ei hoita mälus otsingupuud ja ei vaadata seise kaugemale ette kui jooksva seisu järglased. See tähendab, et igal hetkel vaadatakse ainult seda, kas jooksvast seisust on võimalik ülesmäge edasi minna; kui ei ole, on töö lõpetatud. [6]

Mäkketõus (Algoritmi töö käik) [1]
1. Kontrolli, kas jooksev seis on lõppseis. Kui jah, tagasta jooksev seis ja lõpeta. Lahendus on leitud.

2. Võrdle jooksva seisu hinnangut järglasteks olevate seisude hinnangutega. Kui järglaste hulgas on parema hinnanguga seis, võta see jooksvaks seisuks ja alusta uuesti punktist 1.

3. Kui järglaste hulgas parema hinnanguga seisu ei ole, lõpeta töö ja tagasta jooksev seis. Leitud lokaalne maksimum.

Eelised ja puudused

Kui mäkketõusu algoritm suudab lahenduse leida, leitakse see tavaliselt väga kiiresti. Kui lahendust leida ei ole võimalik, tuleb ka see ruttu välja. Seega töö käigus tehakse vähe samme ning kuna otsingupuud meeles ei hoita, kasutatakse vähe mälu.

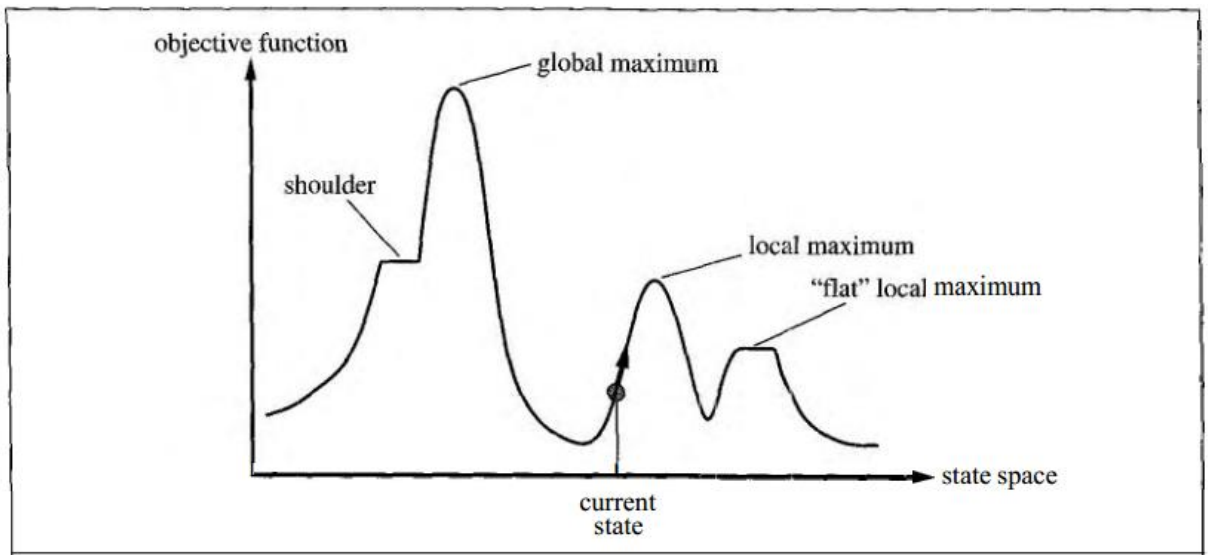
Teiselt poolt aga, kuna liigutakse ainult ülesmäge, juhtub tihti, et lahenduse leidmise asemel jõutakse ühesse järgmistest olukordadest:

- Mäejalam (lokaalne maksimum) – jooksva seisu hinnang on parem kõigist oma järglaste hinnangutest, kuid ei ole tegelikult kõige parema võimaliku hinnanguga seis. Mäestikus liikudes leiti küll mäetipp, kuid on veel kõrgemaid tippe.
- Platoo – olukord, kus võimalike järglaste hulgas ei ole parema hinnanguga seise, kuid on vähemalt üks seis, mille hinnang on sama mis jooksval seisul. Teisel pool platood võib olla parema hinnanguga seis.

[1] Järglasteks on vastavalt eelnevale kokkuleppele, kas ühe või mitme käigu kaugusel olevad seisud. See tähendab, et kui näiteks on eelnevalt kokku lepitud, et järglastena vaadatakse kahe käigu kaugusel olevaid seise, siis koosneb järglaste hulk kõigist sellistest seisudest, mis on saadavad hetkeseisule järjest kahte operaatorit rakendades.

- ❖ Mäeseljak – on erilist tüüpi lokaalne maksimum. See on justkui platoo, millel ükskõik millises suunas liikudes jõutakse seisuni, kust saab edasi liikuda ainult kas sama hinnanguga seisule või allamäge.

Joonisel 1 on kujutatud tasandil olekute ruumi maastikku, kus on välja toodud kõik need olukorrad.



Joonis 1. Olukorrad, kuhu mäkketõusu algoritm sattuda võib – platoo (*shoulder*), mäeseljak („*flat" local maximum*, inglise keeles kasutatakse veel ka väljendit *ridge*), mäejalam (*local maximum*), mäetipp (*global maximum*). [6]

1.2.2. Mäkketõus tagasipöördumisega

Üks võimalus mäkketõusu puudustest lahti saamiseks, on kasutada tagasipöördumist (ingl k *backtracking*). Sellise mäkketõusu algoritmi variandi puhul ei lõpetata lokaalsesse maksimumi jõudes tööd, vaid võetakse samm tagasi. Eelmisest seisust proovitakse siis uuesti ülesmäge jätkata, kuid seis, mille puhul juba kindlaks tehti, et sealt edasi ei saa, jäetakse kõrvale. Kui see seis oli eelmise seisu järglaste hulgas ainuke, mis oli parema hinnanguga, siis võetakse veel üks samm tagasi. Nii jätkatakse, kuni kas leitakse lõpplahendus või jõutakse tagasi algseisu, kuna algseisust sammu tagasi astuda ei saa. [5,7]

Kui tavaline mäkketõus suudab algseisust lahenduse leida, siis toimib ka mäkketõus tagasipöördumisega samamoodi ning leiab sama lahenduse samade sammudega. Erinevused algoritmi töös tulevad sisse alles siis, kui tavaline mäkketõus jääks kinni ning lõpetaks töö ilma lahendust leidmata.

Mäkketõus tagasipöördumisega (Algoritmi töö käik) [5]
1. Kontrolli, kas jooksev seis on lõppseis. Kui jah, tagasta jooksev seis ja lõpeta. Lahendus on leitud.

2. Võrdle jooksva seisu hinnangut veel läbimata järglaste hulgas olevate seisude hinnangutega. Kui nende seisude hulgas on parema hinnanguga seis, võta see jooksvaks seisuks ja alusta uuesti punktist 1.

3. Kui veel läbimata järglaste hulgas parema hinnanguga seisu ei ole ja jooksev seis ei ole algseis, astu samm tagasi ning võta jooksvaks seisuks eelmine seis (jooksva seisu vanem). Mine punkti 2.

4. Kui jõuti tagasi algseisu, see tähendab jooksev seis on algseis, siis lõpeta töö. Lahendust leida ei õnnestunud.

Eelised ja puudused

Kuna lahendus leitakse alati, kui see on võimalik leida tavalise mäkketõusu algoritmiga, ning lisaks on lootus lahendus leida ka siis, kui tavalise mäkketõusuga see ei õnnestu, siis on selge, et tõenäosus lahendust leida, võrreldes tavalise mäkketõusuga, kasvab.

Samas kasvab ka mälukasutus, kuna igal seisul peab olema viit eelmisele seisule, et tagasiliikumine oleks võimalik, ja meeles tuleb hoida ka juba vaadatud seisud, et nendesse uuesti mitte sattuda. Kasvab ka lahenduse leidmise sammude arv. Kui tavaline mäkketõus suudab lahenduse leida, siis on muidugi sammude arv lahenduse leidmisel sama, kuid muudel juhtudel on tehtud vähemalt üks mittevajalik samm, mis on vaja tagasi võtta.

Mäkketõus tagasipöördumisega lõpetab töö ilma lahendust leidmata juhul kui:

- Algseis on platool.
- Algseis on mäeseljak.
- Algseis on lokaalne maksimum või selle ümbruses olev seis, kust parema hinnanguga seisud viivad kõik lokaalsele maksimumile.

1.2.3. Mäkketõus juhusliku uuestialustamisega

Nii tavalise mäkketõusu kui ka tagasipöördumisega mäkketõusu korral ei ole lahenduse leidmine garanteeritud. Lihtsaim viis seda raskust ületada on igal korral, kui algoritm lõpetaks töö lahendust leidmata, alustada uuesti uuest algseisust. Mäkketõus juhusliku uuestialustamisega (ingl k *hill climbing with random restart*) seda põhimõtet järgibki.

Igast algseisust tehakse samad sammud, mis tehtaks tavalise mäkketõusuga, ainuke vahe on, et kui tavaline mäkketõus lõpetaks töö lahendust leidmata, siis mäkketõus juhusliku uuesti-

alustamisega käivitab kogu protsessi otsast peale uue algseisuga. Iga kord, kui satutakse seisu, millest pole võimalik ülesmäge jätkata, võetakse ette uus algseis ja hakatakse otsast peale.
See tähendab, mäkketõus juhusliku uuestialustamisega kasutab korduvalt mäkketõusu algoritmi erinevatel algseisudel, kuni leiab lahenduse, teisisõnu, kuni leitakse algseis, mille tavaline mäkketõusu algoritm suudab lahendada. [6]

Mäkketõus juhusliku uuestialustamisega (Algoritmi töö käik) [6]
1. Kontrolli, kas jooksev seis on lõppseis. Kui jah, tagasta jooksev seis ja lõpeta. Lahendus on leitud.

2. Võrdle jooksva seisu hinnangut võimalike järglaste hinnangutega. Kui nende seisude hulgas on parema hinnanguga seis, võta see jooksvaks seisuks ja alusta uuesti punktist 1.

3. Kui järglaste hulgas parema hinnanguga seisu ei ole, genereeri uus algseis ning võta see jooksvaks seisuks. Alusta uuesti punktist 1.

Eelised ja puudused

Nagu tavalise mäkketõusu puhul, on ka selle algoritmi puhul mälus vaja hoida ainult hetkeseisu ning selle hinnangut. Töö jooksul ei vaadata kaugemale ette kui ühe sammu kaugusel olevad seisud ning ei hoita meeles läbitud tippe.
Erinevalt eelnevalt käsitletud mäkketõusu algoritmi variantidest, leiab mäkketõus juhusliku uuestialustamisega lahenduse (kui see eksisteerib) igal käivitamisel. Seda sellepärast, et igal tavalise mäkketõusu käivitamisel on teatud tõenäosus lahenduseni leida ning käivitamiste arvu suurenedes läheneb see tõenäosus ühele. Näiteks kui tavalise mäkketõusu algoritmiga on lahenduse leidmise tõenäosus 10%, siis tõenäosus, et viie või vähema käivitamisega leitakse ühel korral lahendus, on umbes 40%. Kümne käivitamise korral on see tõenäosus 65%, ning tõenäosus, et leitakse lahendus kahekümne kolme või vähema käivitamisega on rohkem kui 90%. Lisaks sellele, on juba igal algseisu genereerimisel teatud tõenäosus, et algseisuks satub lõppseis ehk lahendus. [6]
Mäkketõus juhusliku uuestialustamisega ei lõpeta kunagi tööd ilma lahendust leidmata.
Samas, mida rohkem kordi juhusliku uuestialustamisega mäkketõus uuest algseisust alustab, seda rohkem samme ka tehakse, kuna iga kord tuleb teha täpselt nii palju samme, kui tavaline mäkketõus vastava algseisu puhul teeks. Näiteks kui tavaline mäkketõus teeb enne töö lõpetamist keskmiselt umbes 10 sammu ning kui lahenduse leidmiseks tuleb uuest algseisust otsast alustada 23 korda, siis on sammude arv umbes 230.

1.2.4. Libalõõmutamine

Kuigi mäkketõus juhusliku uuestialustamisega leiab lahenduse igal korral, võtab lahenduse leidmine väga palju samme kuna igal korral kui kinni jäädakse, genereeritakse lihtsalt uus juhuslik algseis. On selge, et ainult ülesmäge liikudes on lahenduse leidmise tõenäosus üsna väike, on vaja aeg-ajalt teha samm halvema seisu suunas ehk allamäge.

Mäkketõusu algoritm, mis kunagi ei tee sammu halvema seisu suunas, nii öelda allamäge, ei ole täielik, see tähendab, ei suuda igal korral lahendust leida, kuna võib lõpetada töö, sattudes lokaalsele maksimumile. Teiselt poolt aga, kui vaadata eeskirja, mis igal sammul valib järglaste hulgast juhusliku järgmiseks seisuks, siis on tegu küll täieliku algoritmiga, see tähendab lahendus leitakse igal algoritmi kasutamisel, kuid samas väga ebaefektiivse algoritmiga. Tundub mõistlik neid kahte algoritmi, mäkketõusu ja juhuslikkust, mingil viisil kombineerida, et saavutada nii efektiivsus kui ka täielikkus. [6]

Libalõõmutamine (ingl k *simulated annealing*) on üks viis, kuidas kombineerida juhuslikkust ja mäkketõusu algoritmi. Lõõmutamine on metalli töötlemise viis, kus metalli kuumutatakse ning seejärel jahutatakse järk-järgult. Lõõmutamine aitab suurendada metalli vastupidavust ja struktuuri püsivust. Ka libalõõmutamise algoritmis tuuakse sisse mõisted temperatuur ja vähehaaval jahutamine, olenevalt temperatuurist lubatakse aeg-ajalt käike halvemas suunas ehk allamäge.

Kuni võimalik, toimib ka libalõõmutamine justkui mäkketõusu algoritm, kuid nagu ka kahe eelmise algoritmi puhul, tuleb erinevus sisse sel hetkel, kui tavaline mäkketõus lõpetaks töö mingil teisel põhjusel kui lahenduse leidmine. Sellisel juhul tehakse libalõõmutamise korral teatud tõenäosusega samm halvema seisu suunas ehk allamäge. Igal algoritmi sammul, olenemata sellest, kas samm oli tulemuslik või mitte, vähendatakse teatud määral temperatuuri. Mida suurem on temperatuur, seda vastuvõtlikum on algoritm allamäge liikumisele, see tähendab ühtlasi ka seda, et kõrgema temperatuuri puhul võib juhtuda, et liigutakse väga palju halvema seisu suunas, väiksema temperatuuri puhul liigutakse allamäge harvemini ning ka väiksemal määral.

Selle, kui palju temperatuur igal algoritmi sammul väheneb, määrab jahtumiskonstant. Jahtumiskonstant väljendab protsenti, mil määral temperatuuri alandama peaks ning jääb seega vahemikku 0 kuni 1. Temperatuur muutub vastavalt valemile

$$t \coloneqq t * (1 - k),$$

kus t tähistab temperatuuri ning k jahtumiskonstanti.

Kui järglaste hulgas paremat seisu ei ole, võetakse juhuslikult valitud järglane uueks hetkeseisuks ainult siis, kui tõenäosus, et tegu on antud olukorras hea käiguga, on piisavalt suur. See tõenäosus *p* arvutatakse vastavalt valemile

$$p = 1 - \frac{f(juhuslik) - f(jooksev)}{t},$$

kus f on hinnangufunktsioon, *juhuslik* tähistab järglaste hulgast juhuslikult valitud seisu, *jooksev* hetkeseisu ja *t* temperatuuri. Tähelepanu tuleb pöörata sellele, et vastavalt sellele valemile saab tõenäosuse leida siis, kui parema seisu puhul on hinnangufunktsiooni väärtus madalam. Juhul kui paremal seisul on kõrgem hinnang, tuleb lugejas suuruse $f(juhuslik) - f(jooksev)$ asemel kasutada suurust $f(jooksev) - f(juhuslik)$. Selles osas tuleks tähelepanu pöörata sellele, et *p* kui tõenäosus oleks positiivne.

Otsustamiseks, kas tõenäosus on piisavalt suur, on mitu võimalust. Üks variant on eelnevalt määrata, kui suur tõenäosus on piisavalt suur, näiteks kasutada tingimust $p > 0{,}5$. Teine võimalus on igal sammul võtta juhuslik arv vahemikust 0 kuni 1 ning võrrelda tõenäosust sellega.

Kui temperatuur langeb alla mingi piiri, enamasti on selleks 1, siis töö lõpetatakse ka sel juhul, kui lahendust ei leita, sest vastasel juhul hakkaksid mingist hetkest halvema seisu suunas liikumise tõenäosused uuesti suurenema. Teatud arvu sammude järel oleks igasuguse seisu vastuvõtmise tõenäosus juba suurem kui 1.

Libalõõmutamine (Algoritmi töö käik) [1]

1. Kontrolli, kas jooksev seis on lõppseis. Kui jah, tagasta jooksev seis ja lõpeta. Lahendus on leitud.

2. Võrdle jooksva seisu hinnangut järglaste hinnangutega. Kui nende seisude hulgas on parema hinnanguga seis, võta see jooksvaks seisuks, vähenda temperatuuri vastavalt jahtumiskonstandile ja alusta uuesti punktist 1.

3. Kui temperatuur on suurem kui 1 ja võimalike järglaste hulgas parema hinnanguga seisu ei ole, vali nende hulgast juhuslik, arvuta selle seisu vastuvõtmise tõe-näosus. Kui leitud tõenäosus on suurem mingist eelnevalt määratud tõenäosusest, võta seis jooksvaks seisuks, vähenda temperatuuri vastavalt jahtumiskonstandile ja alusta uuesti punktist 1.
Kui leitud tõenäosus on liiga väike, et seisu vastu võtta, siis vähenda temperatuuri vastavalt jahtumiskonstandile ja mine tagasi 3. punkti algusesse.

4. Kui temperatuur on langenud alla ühe, lõpeta töö. Lahendust leida ei ole võimalik.

Puudused ja eelised

Mälus on vaja hoida ainult hetkeseisu, selle hinnangut ning temperatuuri ja jahtumiskonstanti. Töö jooksul ei vaadata vastavatest järglastest (kas siis ühe käigu kaugusel olevad järglased või eelnevalt määratud arvu käikude kaugusel olevad järglased) kaugemale ette ning ei hoita meeles läbitud tippe, see tähendab, et algoritmi tööks on mälus vaja hoida väga vähe.

Kuigi libalõõmutamise puhul on lahenduse leidmise tõenäosus suurem kui tavalise mäkketõusu algoritmi korral, ei ole ka libalõõmutamine siiski täielik. Algoritmi töö käigus langetatakse järk-järgult vastavalt eelnevalt määratud jahtumiskonstandile temperatuuri. Kui algoritmi tööprotsess kestab, lahendust leidmata, nii kaua, et temperatuur langeb alla ühe, siis töö lõpetatakse ilma lahendust leidmata.

Teiselt poolt, võrreldes juhusliku uuestialustamisega mäkketõusuga, on libalõõmutamine allamäge käikude osas natuke kriitilisem ning kaalub iga sellist käiku vastavalt tema hinnangule, seega on libalõõmutamine enamasti võimeline lahenduse leidma kiiremini kui juhuslik uuestialustamine ja on efektiivsem.

See tähendab, et libalõõmutamise puhul on tegemist keskteega tavalise mäkketõusu efektiivsuse ning juhusliku uuestialustamisega mäkketõusu täielikkuse vahel.

2. Probleemide tutvustus

2.1. 8-mäng

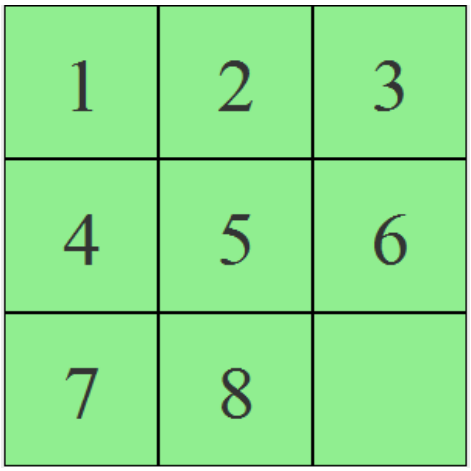
Mängu ajalugu

8-mäng on lihtsam versioon 15-mängust, mis kogus populaarsust 1880. aastate alguses. 15-mängu autoriks peetakse tihti ekslikult ameerika nuputusülesannete koostajat Samuel Loydi. Sellest valearvamusest räägib näiteks Jerry Slocum oma artiklis „Sam Loyd's most successful hoax" [8]. Veel üks mees, keda aeg-ajalt ekslikult autoriks nimetatakse, on Matthias J. Rice. Tegelik autor on aga Noyes Chapman, Canastota postkontori ülem New Yorgist, kelle sõbrad ja tuttavad nägid esialgset versiooni mängust väidetavasti juba aastal 1874. [9]

Suure populaarsuse saavutas pusle aastal 1880, 15-mängule viidati isegi tolle aja popkultuuris – teatrietendustes, luuletustes, muusikas.[9]

Reeglid

Mängulaual on kaheksa numbriruutu, numbritega ühest kaheksani, ning lisaks sellele üks tühi ruut. Tühja ruutu saab vahetada selle all, üleval, vasakul või paremal oleva numbriruuduga. Eesmärgiks on selliste käikudega jõuda olukorrani, kus numbrid ühest kaheksani lähevad järjest ning tühi ruut on kõige lõpus (joonis 1).



Joonis 1. 8-mängu lõppseis.

8-mängu lahendatavus

Iga 8-mängu algseis ei ole lahendatav. Tähelepanu tuleb pöörata inversioonide arvule. Inversiooni permutatsioonis moodustavad kaks elementi, mille puhul permutatsioonis eespool oleva elemendi väärtus on suurem kui permutatsioonis kaugemal asuval elemendil. Näiteks

kui vaadata arvude 1, 2, 3 permutatsioone, siis permutatsioonis 2, 1, 3 moodustavad elemendid 2 ja 1 inversiooni. Permutatsioonis 3, 2, 1 on 3 inversiooni – inversiooni moodustavad paarid (3, 2), (3, 1) ja (2, 1).

8-mängu seisu inversioonide arvu leidmiseks, vaatame seisu kui numbrite 0 kuni 8 permutatsiooni, kus 0 tähistab tühja ruudu asukohta. Kuna tühja ruudu asukoht ei mõjuta lahendatavust, siis elemendi 0 poolt moodustatud inversioone me ei arvesta. See tähendab, et tegelikult on tegu permutatsiooniga elementidest 1 kuni 8, kuid elemendi 0 jätame sisse selleks, et lihtsamini mõista, mida 8-mängu üks lubatud käik permutatsiooni jaoks tähendab.

Kuna 8-mängu lõppseisu puhul lähevad numbrid järjest, on lihtne näha, et inversioonide arv on 0. Olenemata seisust, ei muuda tühja ruudu vasakule või paremale liigutamine (elemendi 0 vahetamine ühega selle kõrval olevatest elementidest) elementide 1 kuni 8 järjestust ning seega jääb ka inversioonide arv sellise käigu puhul samaks.

Tühja ruudu vahetamine selle kohal oleva numbriruuduga tähendab selle numbri (olgu see i) liigutamist permutatsioonis kolm kohta edasi. Kuna ülejäänud elementide positsioonid üksteise suhtes ei muutu, siis eelnevalt eksisteerinud inversioonipaarid, mis ei sisaldanud elementi i, moodustavad ka nüüd inversiooni.

Kuna elementi i liigutati permutatsioonis edasi kolm kohta, siis elementide, mis eelnevalt olid elemendist i eespool, positsioon i suhtes ei muutu. Samuti ei muutu nende elementide positsioon i suhtes, mis olid eelnevalt elemendist i rohkem kui kolm kohta edasi. Seega kõik nende elementide ja elemendi i poolt moodustatud võimalikud inversioonid säilivad. Elemendi i positsioon muutus ainult nende kolme elemendi suhtes, millest ta mööda liigutati. Üks nendest kolmest elemendist on 0 ehk tühi ruut ning tühja ruudu inversioone me ei arvesta. Järelikult on ainult kaks elementi, mille inversioone i suhtes tuleb vaadata.

Esimene võimalik olukord on, et kumbki nendest elementidest ei moodustanud enne elemendiga i inversiooni, sellisel juhul moodustavad nad mõlemad nüüd elemendiga i inversiooni ning inversioonide arv suurenes selle käigu tulemusel kahe võrra.

Teine variant on, et mõlemad elemendid moodustasid enne elemendiga i inversiooni. Sellisel juhul pärast vahetust kadusid mõlemad neist inversioonidest ära ja inversioonide arv vähenes kahe võrra.

Viimane variant on, et üks elementidest moodustas elemendiga i inversiooni, teine mitte. Sellisel juhul nüüd see element, mis enne ei moodustanud inversiooni elemendiga i, nüüd moodustab ning see, mis moodustas, enam ei moodusta. See tähendab, et inversioonide arv jäi samaks.

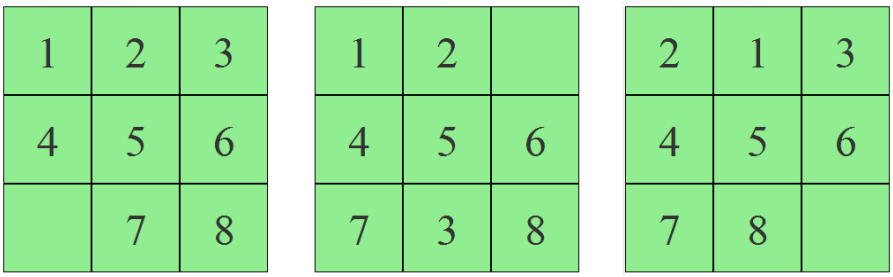
Kokkuvõttes saime, et iga käiguga kas inversioonide arv ei muutu või muutub kahe võrra. Kuna lõppseisu inversioonide arv on 0, siis ilmselt ei ole võimalik lahendada algseise, milles on paaritu arv inversioone.

Optimaalne lahenduskäik

Lahendatava 8-mängu puhul on alati võimalik leida lõppseisuni viiv käikude jada, mis sisaldab maksimaalselt 31 käiku. Siit saame olenemata (lahendatavast) algseisust optimaalse lahenduse ülemiseks piiriks 31 käiku.

Võimalikud heuristikud

8-mängu puhul on võimalikke heuristikuid üsna palju. Üks lihtsamaid on **vales kohas olevate numbriruutude arv**. Sellise heuristiku miinuseks on see, et ta ei sisalda informatsiooni ruutude omavahelise paigutuse kohta ega ka selle kohta, kui lihtne või keeruline oleks ruutu õigele kohale liigutada. Lisaks võivad sama hinnanguga seisud olla tegelikult väga erineval kaugusel lõppseisust. Näiteks saaksid kõik järgmisel joonisel 2 olevad seisud sellist heuristikut kasutades hinnangu 2. Samas on peale vaadates selge, et vasakpoolne seis on lõpplahendusest vaid kahe käigu kaugusel, keskmise seisu lahendust peale vaadates kohe kindlaks määrata ei saa, parempoolne seis on aga koguni lahendamatu.



Joonis 2. Kolm seisu 8-mängus.

Teiselt poolt on aga tegu väga lihtsasti mõistetava ning lihtsasti arvutatava heuristikuga.

Üks levinumaid 8-mängu heuristikuid on **Manhattani kaugus**. [10] Arvutamisel vaadatakse iga ruutu eraldi ning leitakse, mitme koha võrra tuleks seda liigutada, et see oleks lõppseisu suhtes õigel asukohal. Heuristiku väärtuseks mingil seisul on kõigi ruutude kauguste summa. Jällegi on tegu heuristikuga, mis ei sisalda informatsiooni ruutude omavahelise paigutuse kohta, kuid selle asemel, et vastata küsimusele, kas ruut on vales kohas, leitakse, kui palju vales kohas ta on. Samuti on võimalike hinnangute vahemik suurem, mis teeb seisude

omavahelise võrdlemise teatud määral lihtsamaks, kuna sama hinnang omistatakse väiksemale hulgale seisudele.
Ka on tegu lihtsasti arvutatava ja mõistetava heuristikuga.

Üks võimalus Manhattani kaugust täpsemaks teha on kasutada lisaks **lineaarsete konfliktide** arvutamist. [10] Lineaarse konflikti moodustavad kaks numbriruutu siis, kui nad on mõlemad õiges reas või õiges veerus, kuid nende asetus selles reas või veerus on üksteise suhtes vale. Näiteks kui esimeses reas on numbriruudud 2, 7, 1, siis moodustavad ruudud 2 ja 1 lineaarse konflikti, sest nad on mõlemad reas, kus nad peavad olema, kuid ruut 2 on enne ruutu 1, lõppseisus on aga vastupidi numbriruut 1 eespool ruudust 2. Heuristiku väärtuse leidmiseks liidetakse kokku igas reas ja igas veerus olevate lineaarsete konfliktide arvud. Lineaarse konflikti olemasolu lisab Manhattani kaugusele vähemalt kaks käiku, kuna ruute tuleb liigutada üksteise ümber. Seetõttu lisatakse heuristiku arvutamisel Manhattani kauguse väärtusele kahekordne lineaarsete konfliktide arv. Seega kui funktsioon m tähistab Manhattani kaugust ja l lineaarseid konflikte, siis on heuristiline funktsioon

$$h(n) = m(n) + 2 * l(n),$$

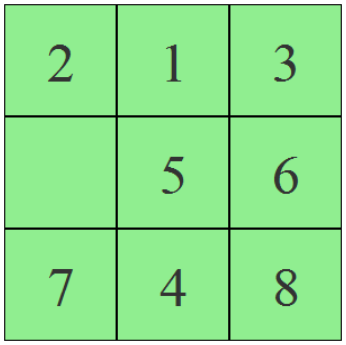
kus n on seis, mille hinnangut arvutatakse.

X-Y heuristik

Nagu öeldud, sisaldavad kolm eelmist heuristikut väga vähe informatsiooni ruutude omavahelise paigutuse kohta. Kui on oluline, et heuristik oleks võimalikult täpne, siis võrreldes eelmistega, on parem variant X-Y heuristik või ka X-Y kaugus. [10]
X-Y heuristiku puhul jagatakse probleem kaheks lihtsamaks osaks. 8-mängu puhul on nendeks osadeks õigetesse veergudesse viimine ja õigetesse ridadesse viimine. Heuristiku väärtus on kahe osaprobleemi lahendamiseks vajalike käikude summa. See tähendab, heuristiku väärtus leitakse nii, et arvutatakse välja minimaalne lubatud käikude arv, mis on vajalik kõigi ruutude õigesse veergu viimiseks, ja arvutatakse vajalik lubatud käikude arv, et kõik ruudud oleksid õiges reas. Liites need kaks kokku saimegi heuristiku väärtuse. Kui x ja y on vastavalt funktsioonid, mis leiavad käikude arvu õigetesse veergudesse ja õigetesse ridadesse liigutamisel, siis on vastav heuristiline funktsioon

$$h(n) = x(n) + y(n),$$

kus n on seis, mille hinnangut arvutatakse.
Näiteks kui seisuks n võtta joonisel 3 kujutatud seis, siis oleks heuristilise funktsiooni väärtus $h(n) = 7 + 2 = 9$.



Joonis 3. 8-mängu algseis, millel X-Y heuristiku väärtus on 9.

X-Y heuristik on täpsem kui Manhattani kaugus, kuid samas on tegemist heuristikuga, mille arvutamine on eelnevatest tunduvalt keerulisem.

2.2. 8 lipu probleem

Mängu ajalugu

8 lipu probleemi püstitas esimesena malemängija Max Bezzel aastal 1848. Probleemi uurisid mitmed erinevad matemaatikud, kuid pikka aega ei suutnud keegi neist täielikku lahendust leida, mis pole üllatav, kui arvestada, et tegu on probleemiga, mida ei saa analüütiliselt lahendada (*a problem that defies analytic solution*).[11] Lahendatud sai probleem esmakordselt aastal 1850, kui Franz Nauck leidis probleemile 12 erinevat lahendust, kasutades katse-eksituse meetodit. Aastal 1874 tõestas Dr S. Gunther, et Nauck oli leidnud kõik võimalikud lahendused, ülejäänud 80 lahendust on neist 12-st tuletatavad sümmeetria põhjal. [10]

Reeglid

8 lipu probleemi puhul on eesmärgiks paigutada malelauale kaheksa lippu nii, et ükski teisele tuld ei annaks. See tähendab igas veerus, reas, diagonaalis võib olla maksimaalselt üks lipp. Kuna veerge ja ridu ongi kaheksa, siis tähendab see ühtlasi, et igas reas ja veerus peab olema täpselt üks lipp.

Algseisuna võib vaadata näiteks tühja malelauda ning käiguna lipu malelauale asetamist, kuid antud töös vaadatakse algseisuna seisu, kus malelauale on igasse veergu juhuslikult asetatud üks lipp. Üheks käiguks loetakse mingi lipu veeru piires ümber paigutamist. Selline lähenemine lihtsustab algoritmi töö visualiseerimist probleemi lahendamisel ning nii on võimalik näha algoritmi tööd erinevate algseisude näitel. Vastasel juhul oleks algseis kogu aeg sama ning juhuslikkust mitte sisaldavate algoritmide puhul tehtaks igal käivitamisel samad sammud.

Võimalikud heuristikud

Tule all olevate lippude arv – iga laual oleva lipu puhul vaadatakse, kas see on tule all või mitte. Heuristiku väärtus mingil seisul vastab sellele, kui mitu lippu on mingi teise lipu poolt tule all. Kuna lauale tuleb asetada kaheksa lippu, siis on sellise heuristilise funktsiooni maksimaalne väärtus 8.

Konfliktsete paaride arv – hinnang antakse vastavalt sellele, mitu paari lippe teineteisele tuld annavad. Näiteks juhul, kui kõik lipud annavad kõigile tuld, näiteks on kõik lipud samas reas või samal diagonaalil, siis on konfliktsete paaride arv 28. See on ühtlasi selle heuristiku suurim võimalik väärtus.

Üleliigsete lippude arv ridades ja diagonaalides – vaadatakse iga rida ja iga diagonaali ning loetakse kokku, mitu lippu rohkem kui üks vastavas reas või diagonaalis on. Kuna üks lipp võib samaaegselt üleliigseks osutuda nii reas kui ka diagonaalis, siis on tegemist tule all olevate lippude arvust erineva heuristikuga. Ka maksimaalne võimalik väärtus on märksa suurem – 16.

See heuristik sisaldab rohkem informatsiooni kui tule all olevate lippude arv ning on samas lihtsamini arvutatav kui konfliktsete paaride arv.

Optimaalne lahenduskäik

Vastavalt eelnevalt kirjeldatud probleemi püstitusele on 8 lipu probleem alati lahendatav. Teades lahendust, saame alati veeru kaupa selles veerus oleva lipu lahenduse suhtes õigele kohale tõsta. Seega on iga algseis lahendatav maksimaalselt 8 käiguga. Optimaalse lahenduse ülemine piir on seega 8 käiku.

3. Programmi ülevaade

Selles peatükis antakse ülevaade bakalaureusetöö osana koostatud programmi võimalustest ja implementatsiooni käigus tehtud valikutest.

3.1. Programmi tutvustus

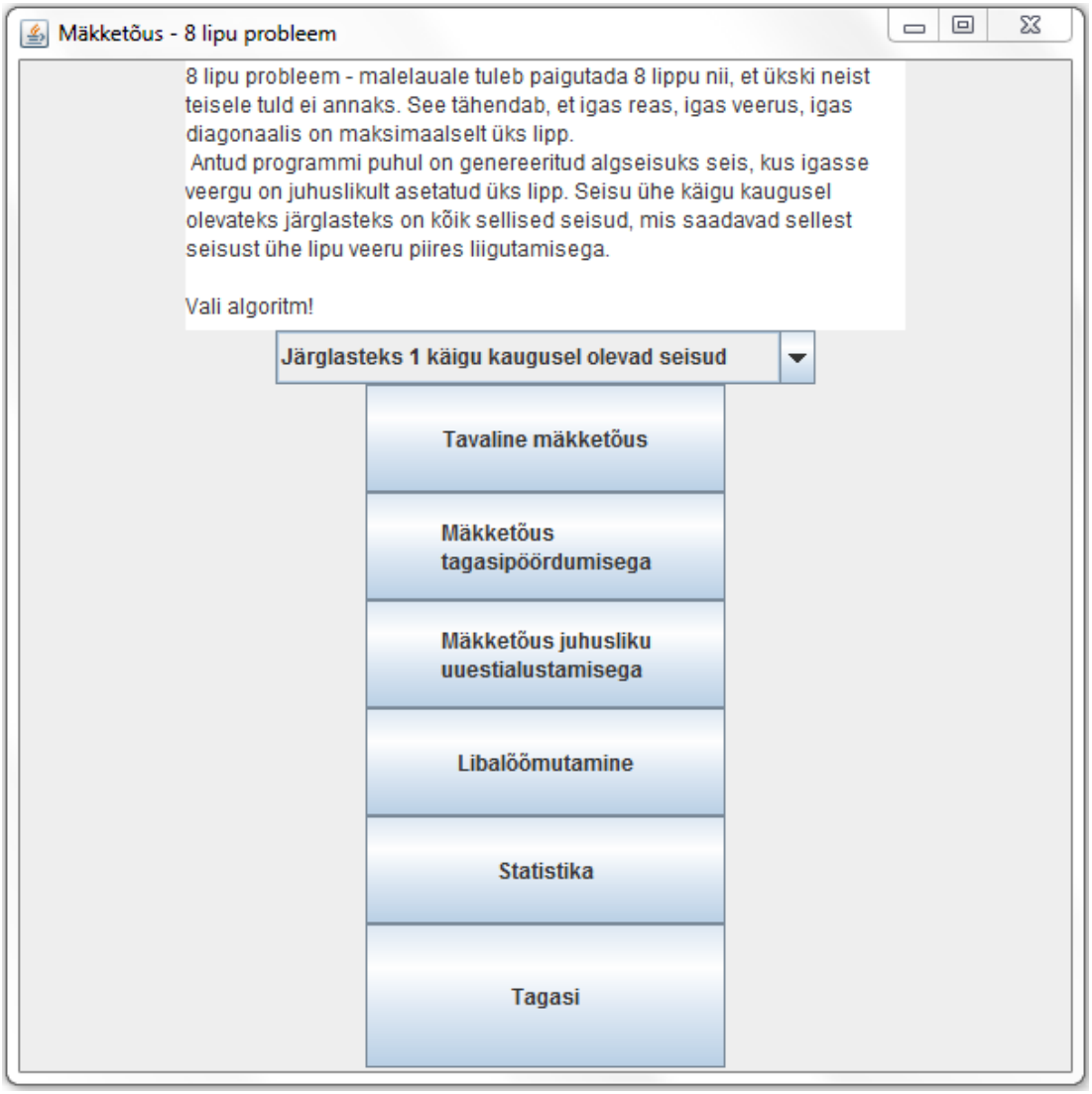
Eesmärk

Programmi eesmärk on visualiseerida mäkketõusu algoritmi ja selle variantide tööd probleemi lahendamisel ja seeläbi lihtsustada algoritmide mõistmist. Samuti annab programm võimaluse erinevaid mäkketõusu algoritmi variante omavahel võrrelda. Omavahel võrrelda saab ka probleeme ning näha, millist mäkketõusu algoritmi varianti milliste omadustega ühe või teise probleemi puhul eelistada.

Võimalused

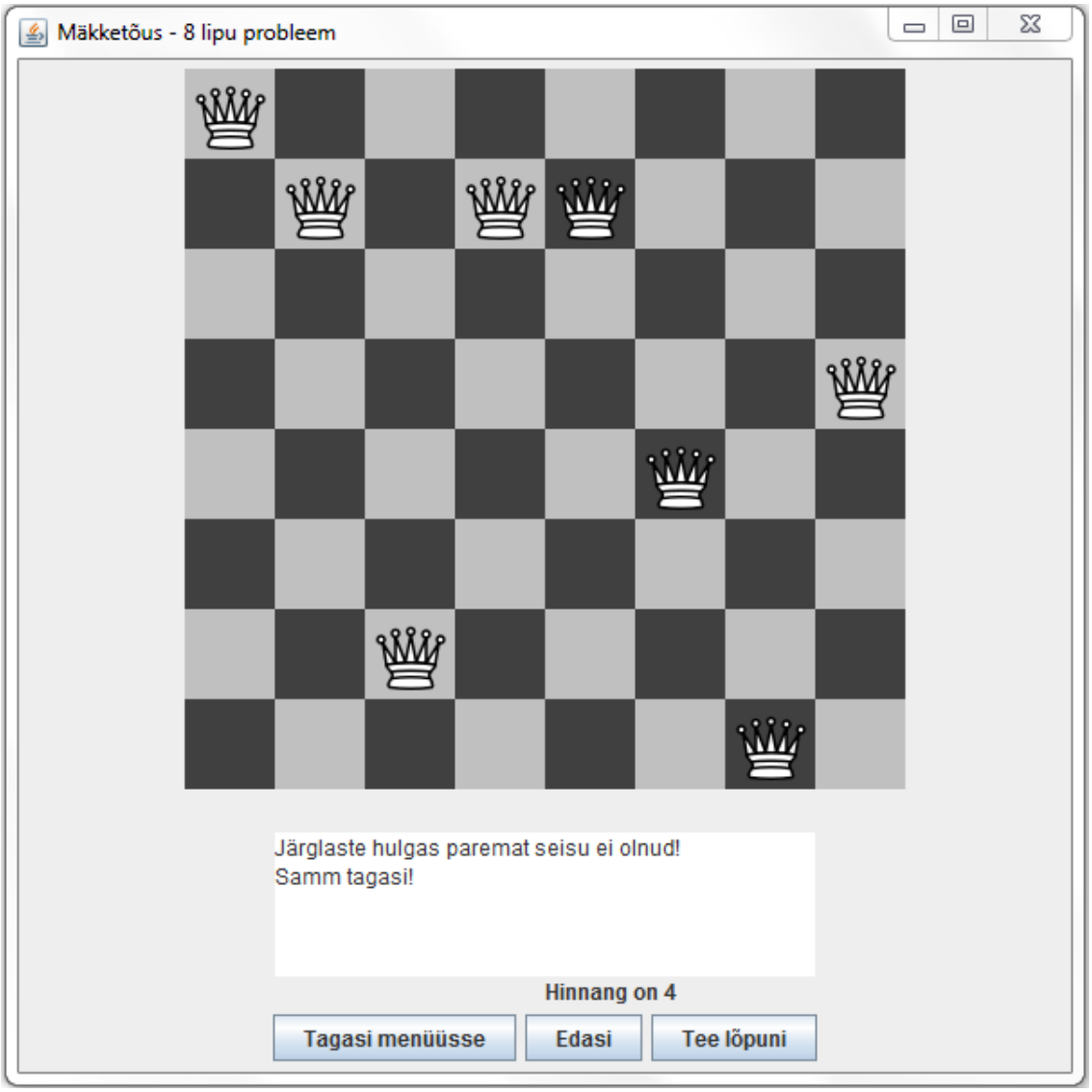
Programm annab kasutajale võimaluse valida kahe probleemi vahel, mille põhjal algoritmidega tutvuda. Olenemata sellest, kas kasutaja valib demonstratsiooni jaoks 8-mängu või 8 lipu probleemi, on edasised valikud samad. Valida saab tavalise mäkketõusu algoritmi, tagasipöördumisega mäkketõusu, juhusliku uuestialustamisega mäkketõusu ja libalõõmutamise vahel.

Lisaks on kasutajal võimalus määrata, mitme käigu kaugusel olevaid seise järglastena vaadatakse. 8 lipu probleemi puhul saab valida ühe või kahe käigu kaugusel olevate seisude vahel. 8-mängu korral on lisaks ühe ja kahe käigu kaugusel olevatele seisudele valikus ka kuue käigu kaugusel olevad seisud. Kuue käigu kaugusel olevad sellepärast, et sellisel juhul on järglaste arv sarnane 8 lipu probleemi korral ühe käigu kaugusel olevate seisude arvuga. See lihtsustab probleemide omavahelist võrdlemist. Programmi aken on näidatud joonisel 4.



Joonis 4. Algoritmi valimise aken, probleemiks on valitud 8 lipu probleem.

Vastavalt kasutaja valikutele esitab programm algoritmi tööd, andes kasutajale selle käigus informatsiooni selle kohta, milline algoritmi samm sooritati ning esitades jooksva seisu graafiliselt. Kuna vahel võib algoritmi töö käigus tehtavate sammude arv olla väga suur, on kasutajale antud võimalus lasta programmil iseseisvalt töö lõpuni teha. Joonisel 5 on esitatud programmi aken, kus tagasipöördumisega mäkketõusu algoritmi töö on pooleli.



Joonis 5. Mäkketõus tagasipöördumisega, algoritmi töö käib. Järgmise sammu nägemiseks tuleb vajutada Edasi nuppu.

Algoritmi töö lõpus annab programm tagasisidet algoritmi edukuse kohta ning salvestab tulemused statistikafaili.

Statistikaga tutvumiseks on pärast probleemi valimist menüüs valik statistika. Statistikast on näha, kui mitu korda mingit algoritmi on kasutatud, kui paljudel kordadel lahendus leida õnnestus ning keskmine sammude ja avatud tippude arv lahenduse leidmisel. Mõlema probleemi ja iga järglaste kauguse valiku jaoks esitatakse statistika eraldi.

3.2. Lisaprogramm statistika kogumiseks

Lisaks põhiprogrammile, on olemas väike lisaprogramm, mis aitab statistika kogumisel. Sellest on kasu, kui soovitakse algoritme omavahel võrrelda, samas ollakse algoritmidega juba tutvunud, ning ei soovita ühe kaupa erinevaid võimalusi läbi proovida ainult selleks, et statistikat saada.

Kasutaja määrab, kui mitme seisu pealt statistikat saada soovib, seejärel genereerib programm kummagi probleemi jaoks vastava arvu algseise ning kasutab igat algoritmi iga võimaliku järglaste valikuga kõigil genereeritud seisudel. Tulemused salvestatakse faili ning kuvatakse ka põhiprogrammiga statistikat vaadates.

Põhjus, miks statistika kogumiseks on tehtud eraldi programm, selle asemel, et lisada põhiprogrammile statistika kogumise funktsioon, on selles, et nii on võimalik kasutada mõlemat paralleelselt. Saab taustal koguda statistikat ning samal ajal tutvuda põhiprogrammi abiga algoritmide tööga. Eelduseks on see, et mõlemad programmid asuvad eraldi kaustas, kuna mõlemad kasutavad samade nimedega statistikafaile.

3.3. Implementeerimise käigus tehtud valikud

8-mängu puhul on võimalikud järglaste valikud 1 käigu, 2 käigu ja 6 käigu kaugusel olevad seisud, 8 lipu probleemi puhul saab valida järglasteks 1 käigu ja 2 käigu kaugusel olevaid seise. Põhjus, miks 8-mängu puhul on valikus lisaks ka 6 käigu kaugusel olevaid seisud, on selles, et 8-mängu puhul on 6 käigu kaugusel olevaid seise olenevalt jooksvast seisust keskmiselt 60. 8 lipu probleemi puhul on ühe käigu kaugusel olevate seisude arv 56. Sarnase järglaste arvu tõttu saab ka kahte probleemi omavahel selles punktis võrrelda ning see annab ka võimaluse jälgida, kui suurel määral valikus olevate järglaste arv tegelikult algoritmi tulemuslikkust mõjutab.

Kasutatavad heuristikud

Sellist tüüpi, algoritmi tööd demonstreeriva programmi jaoks pidasin oluliseks, et iga algoritmi puhul aeg-ajalt lahendus leitakse. Nii on iga algoritmi puhul võimalik näha töö käiku algusest lõpuni: lisaks olukorrale, kus lahendust ei leita, ka olukorras, kus see leitakse. Seda silmas pidades on võetud heuristiku valimise kriteeriumiks, et tavalise mäkketõusu algoritmiga oleks võimalik lahendus leida vähemalt umbes 10% algoritmi kasutamise kordadest.

8 lipu probleemi puhul on heuristikuna kasutatud konfliktsete lipupaaride arvu, see on selliste lipupaaride arv, mis teineteisele tuld annavad. Selline heuristik on võrdlemisi lihtsasti

arvutatav ning sisaldab piisavalt informatsiooni selleks, et ka kõige tavalisema mäkketõusu algoritmiga aeg-ajalt lahendus leida.

8-mängu puhul osutus heuristiku valimine natukene keerulisemaks. Kuna 8-mängu puhul on tipu vahetute järglaste arv väga väike, on hea hinnangufunktsioon võimalikult palju informatsiooni sisaldav ning arvestatava väärtuste vahemiku suurusega, et vähemalt neid väheseid seise omavahel eristada. 8-mängu tutvustava osa juures toodud heuristikutest vales kohas olevate ruutude arvu, Manhattani kauguse ja Manhattani kauguse koos lineaarsete konfliktidega puhul on väga harva järglaste hulgas parema hinnanguga seis, kuna heuristiline funktsioon on vähe informatsiooni sisaldav ja väärtuste vahemik on väike, mis raskendab seisude omavahelist võrdlemist. See aga tähendaks, et tavalise mäkketõusu algoritmiga oleks lahenduse leidmise tõenäosus väga väike ja näiteks juhusliku uuestialustamisega mäkketõusu korral on suurem osa sammudest uue algseisu genereerimine, lootuses, et nüüd genereeritakse lõppseis.

X-Y heuristik aga on antud olukorras liiga keerukas. Kuna X-Y heuristik nõuab minimaalset käikude arvu kummagi alamprobleemi lahendamisel, siis tuleks heuristiku väärtuse arvutamiseks seisul rakendada näiteks A* algoritmi. See aga tähendaks, et heuristilise funktsiooni arvutamine aeglustaks tunduvalt programmi tööd ning sellega kaotaksime ka ühe mäkketõusu algoritmi eelise – lihtsuse.

Pärast mitmete heuristikute katsetamist ja tulemuste analüüsimist, osutus sobivaimaks natukene edasiarendatud versioon Manhattani kaugusest. Iga vales kohas oleva numbriruudu jaoks leitakse lubatud käikude arv, mis on vajalik, et ruut õigele kohale liigutada. See tähendab, et selle asemel, et lihtsalt lugeda, mitme ruudu kaugusel lõppasukohast numbriruut on, loetakse kokku, mitu korda tühja ruutu liigutama peab, et numbriruut oma õigele asukohale jõuaks. Teiste numbriruutude asukohtadele ning nende muutumistele arvutamise jooksul tähelepanu ei pöörata. Heuristilise funktsiooni väärtus leitakse, arvutades järjekorras iga ruudu jaoks õigele kohale viimiseks vajalike sammude arvu ning liidetakse need kokku. See tähendab, et iga ruudu jaoks ei arvutata mitte õigele kohale viimiseks vajalik käikude arv hetkeseisust, vaid sellest seisust, millesse jõuti eelmise ruudu õigele kohale viimise järel. Valem, mille kohaselt heuristiku väärtus arvutatakse, on

$$h(n) = f_1(n) + f_2(n') + f_3(n'') + \cdots + f_8(n''''''''),$$

kus *n* tähistab seisu, millel heuristiku väärtus arvutatakse, *n'* seisu, millesse jõutakse pärast ruudu 1 õigele kohale liigutamist. *n''* on seis kuhu jõutakse seisust *n'* numbriruudu 2 õigele kohale liigutamisel, kusjuures ruut numbriga 1 võib selle käigus oma asukohta muuta, ja nii

edasi. Valemis on f_1, $f_2, f_3, \ldots, f_8$ funktsioonid, mis leiavad käikude arvu vastava numbriruudu õigele kohale liigutamiseks.

Näiteks eelnevalt, 8-mängu tutvustavas osas joonisel 3 kujutatud seisu puhul, oleks ruudu numbriga 1 õigele kohale liigutamiseks vaja kahte käiku. Pärast seda numbriruudu 2 õigele kohale liigutamiseks läheb viis käiku ja nii edasi. Heuristilise funktsiooni väärtus sellel seisul oleks 21.

Võrreldes sellise variandiga, kus iga numbriruudu jaoks arvutatakse õigele kohale viimiseks vajalik käikude arv algseisust, ehk

$$h'(n) = f_1(n) + f_2(n) + \cdots + f_8(n),$$

kus tähistused on samad, mis eelmise valemi puhul, on heuristik, kus arvutamise käigus hetkeseisu tagasi ei minda, kordades efektiivsem. Tabelis 1 on toodud lahendatavuse protsendid mõlemat heuristikut 1000 algseisu korral katsetades, kui järglastena vaadata vahetuid naabreid.

Tabel 1. 8-mängu heuristikute *h(n)* ja *h'(n)* võrdlus: lahendatavuse protsent

	h(n)	***h'(n)***
Tavaline mäkketõus	9,9%	0,2%
Mäkketõus tagasipöördumisega	19,1%	0,3%
Mäkketõus juhusliku uuestialustamisega	100%	100%
Libalõõmutamine	37,7%	7,1%

Kuigi arvutamine on natuke keerulisem kui Manhattani kauguse puhul, on see siiski lihtsam kui X-Y heuristiku puhul. Kui on vaja suuremat väärtuste vahemikku ning täpsust kui Manhattani kauguse puhul, kuid X-Y heuristikuga võrreldes väiksemat keerukust, siis osutub see heuristik üheks sobivaks võimaluseks.

Implementeeritud on ühe ruudu õigele kohale viimise funktsioon nii, et kõigepealt viiakse lubatud käikudega ruut õigesse ritta ning siis õigesse veergu. Selline lähenemine ei anna alati minimaalset käikude arvu, mis ruudu õigele kohale viimiseks vajalik, kuid annab mingi ülemise piiri.

Libalõõmutamise temperatuur ja jahtumiskonstant

Ainuke karakteristik, mis libalõõmutamise ajal pidevalt muutub, on temperatuur. Ülejäänud tegurid - seisude ruum, käigud, hinnangufunktsioon (sama seisu hinnang on sama igal arvutamisel) on eelnevalt defineeritud.[12] Selle tõttu on temperatuur ja jahtumiskonstant libalõõmutamise juures äärmiselt olulised ning mõjutavad suurel määral algoritmi efektiivsust.

Programmis on temperatuuri ja jahtumiskonstandi paarid valitud katse-eksituse meetodil. Kindlasti ei väida ma, et toodud temperatuuri ja jahtumiskonstandi paarid on antud probleemide jaoks parimad, selleks ei ole ma piisavalt palju erinevaid variante läbi proovinud. 8-mängu puhul on valitud algtemperatuuriks 200 ja jahtumiskonstandiks 0,05, ehk 5%. 8 lipu probleemi juures on jahtumiskonstant sama, kuid algtemperatuuriks on valitud 60. 8-mängu puhul on algtemperatuur kõrgem, kuna 8-mängu jaoks valitud heuristiku väärtuste vahemik on suurem. Pikemalt on sellest, miks väärtuste vahemik oluline on, kirjutatud 4. peatükis. Rohkem saab libalõõmutamise võimalikest jahtumisgraafikutest lugeda allikast [12].

4. Tulemuste analüüs

Selle peatüki põhiteemaks on algoritmide efektiivsuse analüüs programmi abiga kogutud statistika alusel. Lisaks selgitatakse, kuidas mingid algoritmi komponendid efektiivsust mõjutavad ja millele ise tehisintellekti algoritme implementeerides tähelepanu peaks pöörama.

1.1. Kogutud statistika

Järgnevalt on toodud programmi abiga kogutud statistika. Statistika kogumiseks on probleemi raames sama algseisu kasutatud iga algoritmi ja iga võimaliku järglaste kauguse korral. Statistika koguti 10 000 genereeritud algseisu põhjal.

Iga mäkketõusu algoritmi puhul on statistika kahes tabelis, vastavalt 8-mängu põhjal kogutud ja 8 lipu probleemi põhjal kogutud statistika.

1.1.1. Tavalise mäkketõusu statistika

Tabel 2. 8-mäng - tavaline mäkketõus

	1 käik	**2 käiku**	**6 käiku**
Lahendatavus	10,19%	14,33%	80,93%
Lahenduse leidmisel			
Samme keskmiselt	23,24	11,29	4,92
Avatud tippe keskmiselt	46,35	43,52	252,46
Lahenduse mitteleidmisel			
Samme keskmiselt	13,55	6,84	2,96
Avatud tippe keskmiselt	27,63	25,97	160,65

Tabel 3. 8 lipu probleem - tavaline mäkketõus

	1 käik	**2 käiku**
Lahendatavus	13,82%	46,03%
Lahenduse leidmisel		
Samme keskmiselt	4,06	2,61
Avatud tippe keskmiselt	207,48	3 408,12
Lahenduse mitteleidmisel		
Samme keskmiselt	3,11	1,99
Avatud tippe keskmiselt	159,7	2 633,74

Esimesena torkab neid tabeleid vaadates silma see, et mida kaugemal olevaid seise järglastena vaadata, seda suurem on lahendatavuse protsent. See tuleneb sellest, et kaugemal olevaid järglasi vaadates on rohkem seise, mille hulgast igal sammul valida. Seetõttu on ka suurem tõenäosus, et nende hulgas on parema hinnanguga seis ja ka suurem tõenäosus, et jõuame neid järglasi mööda liikudes lahenduseni.

Järglaste arvu kasvades töö jooksul tehtavate sammude arv kahaneb, avatud tippude arv aga mitte. Vähem samme on vaja sellepärast, et ühe sammuga tehakse rohkem käike. Samas ühe sammuga ka avatakse rohkem uusi tippe, sest genereeritakse rohkem järglasi. Nendest genereeritud seisudest võetakse jooksvaks seisuks ainult üks, kuid et parimat valida, tuleb genereerida nad kõik. Seetõttu avatud tippude arv koos sammude arvuga ei vähene.

Erandiks on tavaline mäkketõus 8-mängu puhul 1 ja 2 käigu kaugusel olevate järglastega. Tabelis 2 on näha, et neid võrreldes avatud tippude arv väheneb. Põhjuseks on see, et 1 käigu kaugusel olevate järglaste arv ei ole palju väiksem 2 käigu kaugusel olevate arvust. 8-mängu puhul on ühe käigu kaugusel olevaid seise, olenevalt tühja ruudu asukohast, 2-4. Kahe käigu kaugusel olevaid erinevaid järglasi on vastavalt 5-9. Tähele tuleb panna, et igat seisu ehk tippu avatakse ühe korra. Samasse seisu on alati rohkem kui üks tee, kuid kui tipp on kord juba avatud, siis järgmine kord, kui see genereeritakse, siis seda uuesti avatud tippude hulka ei lisata, sest see on seal juba olemas. Kahe käigu kaugusel olevaid järglaseid vaadates tehakse siiski kaks käiku korraga ning seega jõutakse lahenduseni väiksema arvu sammudega, kuna aga järglaste arvu ehk igal sammul avatud tippude arvu erinevus ei ole nii suur, siis avatud tippude arv ei suurene nii, nagu muudel juhtudel, kui kaugemal olevaid järglasi vaadata.

Tavalise mäkketõusu puhul tehakse lahenduse mitteleidmisel vähem samme kui lahenduse leidmisel. Lahenduse leidmiseks tuleb teha iga samm parema hinnanguga seisu suunas, kuni leitakse lõppseis. Kui mingil sammul ei ole võimalik parema hinnanguga seisu jooksvaks seisuks võtta, siis töö lõpetatakse koheselt lahenduse mitteleidmisega. Tõenäosus, et järgmise sammuga leitakse lahendus, igal sammul kasvab, kuid igal sammul on ka teatud tõenäosus, et jooksvast seisust edasi ei saa liikuda. Kuna tõenäosus, et järgmisel sammul edasi ei saa minna, on töö alguses suurem kui tõenäosus, et järgmisel sammul leitakse lahendus, siis enamasti, kui lahendust leida ei õnnestu, tehakse samme vähem kui lahenduse leidmisel.

8-mängu puhul 6 käigu kaugusel olevate järglaste arv on sarnane 8 lipu probleemi puhul 1 käigu kaugusel olevate seisude arvuga. Kui omavahel võrrelda tabelis 2 kuue käigu veergu ja tabelis 3 ühe käigu veergu, siis on näha, et sammude arvud ja avatud tippude arvud on üsna sarnased. Lahendatavuse protsendid aga on üsna erinevad. Põhjuseks on ilmselt see, et 8-

mängu korral on kasutatud paremat heuristikut. See tähendab heuristikut, mis on suurema väärtuste vahemikuga ja täpsem.

Tavaline mäkketõusu algoritm sobib väga hästi siis, kui oluline on see, et algoritm lõpetaks töö võimalikult väikese arvu sammudega, see aga, kui tihti lahend leitakse, ei ole nii oluline.

1.1.2. Mäkketõus tagasipöördumisega: statistika

Tabel 4. 8 mäng – mäkketõus tagasipöördumisega

	1 käik	2 käiku	6 käiku
Lahendatavus	19,16%	41,78%	99,78%
Lahenduse leidmisel			
Samme keskmiselt	70,31	61,12	11,63
Avatud tippe keskmiselt	92,1	122,34	295,67
Lahenduse mitteleidmisel			
Samme keskmiselt	91,17	61,56	12,43
Avatud tippe keskmiselt	86,86	96,31	157,04

Tabel 5. 8 lipu probleem – mäkketõus tagasipöördumisega

	1 käik	2 käiku
Lahendatavus	95,5%	99,95%
Lahenduse leidmisel		
Samme keskmiselt	49,36	14,45
Avatud tippe keskmiselt	1 172,29	6 497,78
Lahenduse mitteleidmisel		
Samme keskmiselt	110,89	7,5
Avatud tippe keskmiselt	2 287,7	2 800,5

Kuna mäkketõus tagasipöördumisega töötab suuremas osas samal põhimõttel kui tavaline mäkketõus, siis ka selle algoritmi puhul tähendab suurem järglaste arv suuremat tõenäosust lahendus leida. Samadel põhjustel, nagu ka tavalise mäkketõusu puhul, väheneb kaugemal olevaid järglasi vaadates töö jooksul tehtud sammude arv ja suureneb avatud tippude arv.

Erinevalt tavalisest mäkketõusust aga tehakse lahenduse mitteleidmisel rohkem samme kui lahenduse leidmisel. See on tingitud sellest, et tagasipöördumisega mäkketõusu puhul on tingimus, millel töö lõpetatakse ilma lahendust leidmata, erinev. Töö lõpetatakse ainult juhul, kui jõutakse tagasi algseisu ning ei ole enam parema hinnanguga järglasi, mida proovitud ei

oleks. See tähendab, et selleks, et algoritm lõpetaks töö, tuleb iga edasi sammu jaoks teha ka samm tagasi. Tabelis 5 paistab, nagu see ei kehtiks 8 lipu probleemi ja 2 käigu kaugusel olevate järglaste korral, kuid põhjuseks, miks see statistikas välja ei tule, on see, et 8 lipu probleemi korral on kahe käigu kaugusel olevaid järglasi vaadates tagasipöördumisega mäkketõusu lahendatavuse protsent 99,95%. See tähendab, et kuna statistika koguti 10 000 algseisu põhjal, siis lahenduse mitteleidmise korral on statistika arvutatud ainult 5 algseisu ehk algoritmi 5 kasutamise korra pealt. Kuna arvutamisel on arvestatud keskmised nii väikese koguse andmete pealt, siis sisaldab see osa statistikast väga vähe informatsiooni tegelike keskmiste kohta.

Kui vaadata tehtud sammude ja avatud tippude suhet, siis on näha, et mäkketõus tagasipöördumisega avab sama arvu sammude korral vähem tippe. Põhjuseks on see, et mingi osa sammudest tagasipöördumisega mäkketõusu puhul on sammud tagasi. Tagasi sammudel uusi tippe ei avata, kuna selles seisus on juba mingil varasemal hetkel oldud ja kõik järglased on juba korra genereeritud.

Mäkketõus tagasipöördumisega töötab 8 lipu probleemi puhul väga hästi, kuna võimalikke lõppseise on palju, järglaste arv on suur (1 käigu kaugusel olevaid järglasi on 56, 2 käigu kaugusel olevaid ~1500) ning igast algseisust on ideaalis võimalik leida lahendus vähem kui 8 käiguga.

8-mängu puhul on lahendatavuse protsendid natuke madalamad, kuid siiski märgatavalt kõrgemad kui tavalise mäkketõusu korral.

1.1.3. Mäkketõus juhusliku uuestialustamisega: statistika

Tabel 6. 8-mäng – mäkketõus juhusliku uuestialustamisega

	1 käik	**2 käiku**	**6 käiku**
Lahendatavus	100%	100%	100%
Lahenduse leidmisel			
Samme keskmiselt	155,52	59,89	6,55
Avatud tippe keskmiselt	313,94	220,4	302,78
Lahenduse mitteleidmisel			
Samme keskmiselt	0	0	0
Avatud tippe keskmiselt	0	0	0

Tabel 7. 8 lipu probleem – mäkketõus juhusliku uuestialustamisega

	1 käik	2 käiku
Lahendatavus	100%	100%
Lahenduse leidmisel		
Samme keskmiselt	34,65	7,36
Avatud tippe keskmiselt	1 515,55	7 918,98
Lahenduse mitteleidmisel		
Samme keskmiselt	0	0
Avatud tippe keskmiselt	0	0

Mäkketõus juhusliku uuestialustamisega on täielik algoritm ja lõpetab töö ainult lahenduse leidmisel, seetõttu ei ole ka statistikat lahenduse mitteleidmise kohta.

Jätkuvalt kehtib see, et kaugemal olevaid järglasi vaadates on lahendatavuse protsent kõrgem. Kuigi kahe käigu kaugusel olevaid seise järglastena vaadates tehakse ühe sammuga 2 käiku, on lahenduse leidmiseks tehtud käikude arv rohkem kui kaks korda väiksem kui 1 käigu kaugusel olevate järglaste puhul. Samamoodi on 6 käigu kaugusel olevate järglaste korral lahenduse leidmiseks tehtud sammude arv rohkem kui 6 korda väiksem, 8-mängu puhul tabeli 6 põhjal isegi umbes 23 korda väiksem. Põhjuseks on see, et mäkketõus juhusliku uuestialustamisega toetub ikkagi tavalisele mäkketõusule. Tabeli 2 põhjal on 1 käigu kaugusel olevaid järglasi vaadates tavalise mäkketõusu lahendatavuse protsent 10,19%, see tähendab igal juhuslikul uuestialustamisel on tõenäosus 10,19%, et seekord leitakse lahendus. 6 käigu kaugusel olevate järglaste korral on vastav protsent tabeli 2 põhjal 80,93%. See tähendab, et kaugemal olevaid järglasi vaadates on enamasti vaja vähem kordi juhuslikust kohast uuesti alustada ja tehakse ka selle võrra vähem samme. Muidugi mõjub ka see, mitu käiku korraga ühe sammuga tehakse, aga tavalise mäkketõusu lahendatavuse tõus vähendab vajalike sammude arvu veelgi.

Kui oluline on täielikkus, siis on mäkketõus juhusliku uuestialustamisega ainus sobiv algoritm vaadatud mäkketõusu algoritmidest.

1.1.4. Libalõõmutamise statistika

Tabel 8. 8-mäng – libalõõmutamine

	1 käik	2 käiku	6 käiku
Lahendatavus	38,45%	77,95%	99,98%
Lahenduse leidmisel			
Samme keskmiselt	42,72	32,04	7,53
Avatud tippe keskmiselt	80,97	101,43	296,8
Lahenduse mitteleidmisel			
Samme keskmiselt	59,49	53,52	28,67
Avatud tippe keskmiselt	112,12	166,56	755

Tabel 9. 8 lipu probleem - libalõõmutamine

	1 käik	2 käiku
Lahendatavus	87,45%	99,92%
Lahenduse leidmisel		
Samme keskmiselt	21,17	8,21
Avatud tippe keskmiselt	906,36	7 971,24
Lahenduse mitteleidmisel		
Samme keskmiselt	49,15	35,33
Avatud tippe keskmiselt	2 296,72	39 607,11

Ka libalõõmutamise puhul kehtib see, et mida kaugemal olevaid järglasi vaadata, seda suurem on lahendatavuse protsent ja seda väiksem sammude arv. Põhjused on samad, mis eelnevalt vaadatud algoritmide puhul.

Lahenduse mitteleidmisel tehakse rohkem samme kui lahenduse leidmisel, kuna libalõõmutamine lõpetab töö lahendust leidmata juhul, kui algoritmi töö on kestnud nii kaua, et temperatuur on langenud alla ühe. See tähendab, et lahendus leitakse ainult siis, kui see on võimalik leida enne algoritmi tööga nii kaugele jõudmist. Seetõttu tehaksegi lahenduse leidmise korral samme vähem kui lõpetamisel põhjusega, et temperatuur on liiga madal, et jätkata.

Kui mäkketõus juhusliku uuestialustamisega kui täielik algoritm kõrvale jätta, siis on libalõõmutamise korral lahendatavuse protsent kõrgeim. Võrreldes juhusliku uuestialustamisega mäkketõusu ja tagasipöördumisega mäkketõusuga, on libalõõmutamine kiirem, see tähendab algoritmi töö käigus tehakse vähem samme.

Kui algoritmi kiirus on olulisem kui täielikkus, kuid on siiski ka vaja võimalikult tihti lahendus leida, siis on libalõõmutamine vaadatud mäkketõusu algoritmi variantidest sobivaim.

1.2. Millele tähelepanu pöörata

Heuristik

Heuristiku hindamise puhul tuleb jälgida mitmeid tegureid, nagu täpsus, keerukus, väärtuste vahemik. Heuristiku abiga määratakse, millise seisu suunas liikuda, milline seis on parem. Seega on ülimalt oluline, et hinnangufunktsioon annaks paremale seisule ka parema hinnangu. See tähendab, et lõppseisule lähemal olev seis peaks saama ka parema hinnangu, et algoritmi abiga lõppseisuni jõudmine oleks võimalik. Kui asjale nii läheneda, võib mõelda, et kõige täpsem heuristik on ju siis selline, mis täpselt määrab sammude arvu, mis on vaja teha, et sellest seisust lõppseisu jõuda. See aga tähendaks, et lihtsalt heuristiku arvutamiseks tuleks iga järglaste seas oleva seisu jaoks leida tee lõppseisuni ning lugeda sammud, mis selle käigus tehakse. See aga tähendab, et hinnangute leidmise käigus leitakse juba korduvalt lahendus. Ilmselgelt ei ole see kasulik.

Seega heuristik peaks küll olema võimalikult täpne, kuid samas ka võimalikult lihtne. Üldiselt oleks hea, kui hinnangufunktsiooni arvutamine ei oleks oluliselt keerukam kui kogu algoritm ise.

Veel tuleks hinnangufunktsiooni valimisel tähelepanu pöörata väärtuste vahemikule. Mida vähem erinevaid väärtusi hinnangufunktsioon omandab, seda rohkematele seisudele vastab üks ja sama väärtus. Ühe ja sama hinnanguga seisud on aga sellised seisud, millel algoritm ei oska vahet teha. Muidugi on ka selliseid seise, mis tõesti on sama head või halvad, millest lõppseisuni on sama arv samme, kuid enamasti tähendab see seda, et sama hinnanguga seisude hulgast võib osutuda valituks selline, mis pikendab teed lahendini.

Libalõõmutamise temperatuur ja jahtumiskonstant

Valimisel tuleb suurt tähelepanu pöörata heuristikule, eriti heuristiku võimalikele väärtustele ning väärtuste vahemiku pikkusele. Libalõõmutamise temperatuuri ja jahtumiskonstanti tuleb alati vaadata koos, eraldi sisaldavad nad väga vähe informatsiooni näiteks selle kohta, kui palju mittetulemuslikke samme lubatakse, see tähendab neid samme, kus mingit seisu kaalutakse, kuid vastu ei võeta.

Temperatuuri valimisel tuleb suurt tähelepanu pöörata hinnangufunktsiooni väärtuste vahemiku suurusele. Kuna tõenäosus, millega libalõõmutamine lubab käigu halvema seisu suunas, leitakse vastavalt valemile

$$p = 1 - \frac{f(juhuslik) - f(jooksev)}{t},$$

siis mida suurem on võimalike hinnangute vahemik, seda suurema väärtuse võib nimetaja, $f(juhuslik) - f(jooksev)$, saada, ning kui temperatuur on määratud liiga madal, siis võib juba esimesel käigul juhtuda selline olukord, kus

$$\frac{f(juhuslik) - f(jooksev)}{t} > 1$$

ja sellest tulenevalt $p < 0$, mis tähendab, et algoritm ei saa tööd jätkata.

Kui väikese hinnangufunktsiooni väärtuste vahemiku puhul valida väga suur temperatuur ja väga väike jahtumiskonstant, siis ei erine libalõõmutamine eriliselt sellisest mäkketõusu algoritmi variandist, mille korral igal kinnijäämisel valitakse järgmine seis järglaste hulgast juhuslikult.

Näiteks oletame, et hinnangufunktsioon omandab täisarvulisi väärtusi vahemikus null kuni viis, kus hinnang null vastab lõppseisule. Hinnanguga 0 seisu leidmisel on algoritmi töö lõppenud. Oletame nüüd, et hetkeseisul on hinnang 1 ning järglaste hulgas paremat seisu ei ole, see tähendab järglaste võimalikud hinnangud on 1 kuni 5. Oletame, et temperatuur on kõrge, näiteks 100, ning jahtumiskonstant väga väike, näiteks 0,005. Kõige halvem käik, mida teha saame, on käik seisu suunas, mis on hinnanguga 5. Sellisel juhul on tõenäosus, et käik tehakse, 1 - 4/100 = 0,96. Ehk siis käik võetakse suure tõenäosusega vastu. Algoritmi töö alguses ei ole sellest mitte mingit probleemi, libalõõmutamine võtabki alguses ka väga halva käigu vastu suurema tõenäosusega kui algoritmi töö lõpu poole. Kuna aga jahtumiskonstant on nii väike, siis on ka 20 käiku hiljem temperatuur üle 90 ning tõenäosus, et väga heast seisust tehakse samm kõige halvemasse suunda, on ikka veel rohkem kui 0,95.

Ehk siis sellisel juhul ütleb tõenäosus väga vähe selle kohta, kui palju halvemaks seis läheb. See aga tähendab, et käike tehakse üsna juhuslikult.

Samas on kõrge temperatuur ja väike jahtumiskonstant õigustatud, kui heuristiku väärtuste vahemik on väga suur. Kui näiteks hinnangute vahe võib kõikuda tuhandetes, siis tekiks nii madala temperatuuriga kui 100 üsna ruttu olukord, kus temperatuur on madalam kui väärtuste vahemiku laius ja tõenäosus tuleb negatiivne, $p < 0$, mis tähendab kohest töö lõppu.

Väike jahtumiskonstant tuleb valida, kui on suur tõenäosus, et enne käigu tegemist proovitakse alati läbi suur hulk erinevaid seise. Tuleb arvestada, et mida suurem jahtumiskonstant, seda vähem käike lubatakse, enne kui otsustatakse, et lahenduse leidmine ei ole võimalik.

Järglaste arv

Järglasteks olevate seisude arv mõjutab otsingu efektiivsust. Piltlikult võib sellest mõelda nii, et järglaste arv on suundade arv, millest mäetipule on võimalik läheneda. Mida rohkem on järglasi, seda suurem on tõenäosus ühele nendest suundadest sattuda. Ka tähendab suurem järglaste hulk seda, et meil on suurem hulk tippe, mille seast parimat järgmist käiku valida.

Enamasti on probleemi puhul olemas üks loogiline viis, kuidas järglased leitakse, kuid kohati annab nende arvu siiski ka mõjutada. Näiteks võib vaadata ühe käiguga saadavate seisude asemel järglastena seise, mis on kahe käigu kaugusel.

Kui järglaste arv on väga väike, näiteks kui igal seisul on ainult kaks-kolm järglast ja ka hinnangufunktsiooni väärtuste vahemik ei ole väga varieeruv, siis on tõenäosus, et järglaste hulgas leidub parema hinnanguga seis, üsna väike. Seega võib ju mõelda, et mida rohkem järglasi, seda suurem on tõenäosus, et nende hulgas leidub seis, mis viib lõppseisule sammu võrra lähemale. Kuid näiteks võib mõelda olukorrale, kus järglasteks võtta absoluutselt kõik võimalikud seisud, sellisel juhul on lõppseis kindlasti järglaste hulgas, kuid tuleb genereerida kõik võimalikud seisud ning siis nende hulgast lõppseis otsida. Selline tegutsemisviis on aga tihti ka väiksemate probleemide korral süsteemile kurnav ning lisaks sellele kaotab algoritmi kasutamine täielikult oma mõtte.

Järglaste arvu puhul tuleb tähele panna, et mingist hetkest ei tasu nende juurde lisamine ennast enam ära. Seega liiga vähe järglasi vähendab lahenduse leidmise tõenäosust või vähemalt suurendab sammude arvu, mis selleks tegema peab. Liiga palju järglasi tingib selle, et vaadatakse läbi liiga palju seise, mis teeb lahenduse leidmise aeglasemaks ja suurendab tunduvalt ressursside kulu.

Parima järglaste arvu valimisel tuleb ka arvestada hinnangufunktsiooni väärtuste vahemikku.

Võimalike lahendite arv

Tähelepanu tuleks pöörata ka sellele, kas ülesandel leidub ainult üks lahendus, üks seis, mida loetakse lõppseisuks, või on neid mitu. Esialgu ei tundu see võib-olla nii ilmne, kuid kui kõrgeimaid tippe on mitu, on ju suurem tõenäosus sattuda sellisele mäele, mida pidi otse üles liikudes leitaksegi kõrgeim tipp.

5. Võimalikud edasiarendused

Mäkketõusu algoritmi ja selle variante realiseeriva programmi üheks võimalikuks edasiarenduseks on anda kasutajale lisaks võimalus proovida erinevaid libalõõmutamise temperatuuri ja jahtumiskonstandi paare. Põhjus, miks seda siin realiseeritud ei ole, on see, et sarnase põhimõttega rakendus, kus saab proovida erinevaid libalõõmutamise näitajaid 8 lipu probleemi lahendamisel, on internetis kättesaadav. [13] Siiski annaks see lisandus kindlasti midagi juurde, kui ka võimaldada erinevate temperatuuri ja jahtumiskonstandi paaride omavaheline võrdlemine statistika kogumise abil. Selle kõrval aga on oht, et programm muutub liiga keeruliseks, kui sundida kasutajat veel mingeid valikuid tegema.

Statistika kogumine programmi abiga on praegu aeglane ning seda ka abiprogrammi kasutades. Suure hulga algseisude pealt, näiteks 10 000 algseisu korral, võib statistika kogumine võtta mitu tundi. Programmi võib kiiremaks teha näiteks jagades statistika kogumist mitme lõime vahel või mõnel muul viisil. Igal juhul leidub mooduseid statistika kogumist kiirendada.

Kokkuvõte

Mäkketõusu algoritm on heuristilise otsingu algoritm, mille peamiseks kitsaskohaks on takerdumine lokaalsetesse maksimumidesse. Selle asjaolu mõju vähendamiseks kasutatakse erinevaid, vähemal või suuremal määral modifitseeritud variante mäkketõusu algoritmist, millest igaühe puhul tehakse mingi kompromiss täielikkuse ja efektiivsuse vahel. Näiteks on juhusliku uuestialustamisega mäkketõus täielik, kuid lahenduse leidmiseks tuleb teha palju samme.

Antud töö eesmärgiks on selgitada mäkketõusu algoritmi ja selle variantide tööpõhimõtet ning illustreerida töö käiku ja seisude vahel liikumist programmi abiga.

Töös on selgitatud otsingualgoritmidega seotud mõisteid ning toodud vaadeldavate algoritmide (mäkketõusu algoritm, mäkketõus tagasipöördumisega, mäkketõus juhusliku uuestialustamisega, libalõõmutamine) eeskirjad ja lühidalt räägitud nende eelistest ja puudustest. Lisaks on käsitletud kahte võimalikku probleemi, 8-mäng ja 8 lipu probleem, mille lahendamisel võib kasutada mäkketõusu algoritmi. Mõlema probleemi juures on räägitud ka võimalikest heuristikutest.

Lõputöö tulemusena valmis programm, mis tutvustab erinevate mäkketõusu algoritmi variantide tööpõhimõtet 8-mängu ja 8 lipu probleemi näitel, näidates seisude vahelist liikumist algoritmi töö käigus. Programm salvestab ka igal algoritmi kasutamisel statistikat selle kohta, kas lahendus leiti ning kui palju töö käigus samme tehti ning tippe avati.

Töö analüüsivas osas on programmi abiga kogutud statistika põhjal toodud välja erinevused algoritmide efektiivsuses. Lisaks on toodud välja ja selgitatud tegureid, mis algoritmi efektiivsust mõjutavad.

Töö sobib kasutamiseks lisamaterjalina kursusel Tehisintellekt I.

Kirjandus

[1] M. Koit, T. Roosmaa. *Tehisintellekt*, TÜ Kirjastus, 2011.

[2] D. Bogdanov. *Optimaalse teekonna leidmise ülesande lahendamine geomeetriliste objektide puude abil*, semestritöö, Tartu Ülikool, 2004.

[3] D. Kopec, T.A. Marsland. *Artificial Intelligence: Search Methods*, 2001. http://webdocs.cs.ualberta.ca/~tony/RecentPapers/Draft.5.2.pdf – viimati vaadatud 6.05.2014.

[4] G. Gigerenzer, R. Hertwig T. Pachur *Heuristics: The Foundations of Adaptive Behavior*, Oxford University Press, 2011.

[5] Algoritmide koostamise strateegiad http://www.cs.tlu.ee/~inga/alg_andm/strategies.pdf - viimati vaadatud 6.05.2014.

[6] S. Russell, P. Norvig *Artificial Intelligence: A modern approach*, Pearson Education, 2rd ed., 2003.

[7] E. Rich, K. Knight, S. Nair *Artificial Intelligence*, McGraw-Hill, 3rd ed., 2009.

[8] J. Slocum. *Sam Loyd's Most Successful Hoax*. http://www.indiana.edu/~liblilly/collections/overview/puzzle_docs/Sam_Loyd_Succesful_Hoax.pdf - viimati vaadatud 6.05.2014.

[9] A. Archer. Rewiev of J. Slocum, D. Sonneveld *The 15 Puzzle: How it Drove the World Crazy* http://isites.harvard.edu/fs/docs/icb.topic446952.files/TheFifteenPuzzle.pdf - viimati vaadatud 6.05.2014.

[10] Heuristics wiki – Heuristics of N-Puzzle http://heuristicswiki.wikispaces.com/N+-+Puzzle - viimati vaadatud 6.05.2014.

[11] B. Abramson ja M. Yung *Construction Through Decomposition: A Linear Time Algorithm for the N-queens Problem*, Department of Computer Science, Columbia University, 1984.

[12] Y. Nouraniy, B. Andresenz. *A comparison of simulated annealing cooling strategies*, IOP Publishing Ltd, 1998.

[13] F. Soto. *Simulated Annealing and the 8-Queen Problem*, Sample implementation. http://ebobby.org/2012/10/22/SimulatedAnnealingEightQueenProblem.html- viimati vaadatud 6.05.2014

Lisad

Lisa 1 - Mäkketõusu algoritmi programm

Töö käigus loodud programm, mille eesmärk visualiseerida mäkketõusu algoritmi ja selle variantide tööd. Käivitamiseks on vajalik Java olemasolu (versioon 1.7 või hilisem).

Lisa 2 - Statistika kogumise programm

Abiprogramm, mis kasutab mäkketõusu algoritmi programmi, et koguda algoritmide kohta statistikat. Ka selle käivitamiseks on vajalik Java (versioon 1.7 või hilisem).

Lisa 3 – Programmide kasutusjuhend

Mäkketõusu programmi ja statistika kogumise programmi funktsioone ja nende kasutamist kirjeldav juhend.

Lisa 4 – Lähtekood

Lisa 5 – Statistikafailid

Programmi abiga genereeritud statistikafailid, mille põhjal on tehtud töö analüüsiv osa. Statistikafailide vaatamiseks võib kasutada tekstiredaktorit või panna failid samasse kausta mäkketõusu programmiga ja vaadata statistikat programmi abiga.

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks

Mina, Mari-Liis Oldja
(sünnikuupäev: 15.07.1992)

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) enda loodud teose
   Mäkketõusu algoritm ja selle variandid kahe probleemi näitel

mille juhendaja on Mare Koit,

1.1. reprodutseerimiseks säilitamise ja üldsusele kättesaadavaks tegemise eesmärgil, sealhulgas digitaalarhiivi DSpace-is lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
1.2. üldsusele kättesaadavaks tegemiseks Tartu Ülikooli veebikeskkonna kaudu, sealhulgas digitaalarhiivi DSpace´i kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.

2. olen teadlik, et punktis 1 nimetatud õigused jäävad alles ka autorile.

3. kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest tulenevaid õigusi.

Tartus, **13.05.2014**