

Mitä testattiin ja miten

Ohjelman testit ovat painottuneet lähinnä JUnit-tyyppisiin testeihin, joissa tärkeää on ollut muun muassa tehokkuus, mutta myös algoritmien oikeellisuus. Käytännössä kaikki luokat ja metodit, jotka ovat tärkeitä algoritmin toimimiselle, on testattu. Testit ovat toteutettu palasissa, jolloin on saatu hyvät tulokset algoritmin oikeellisuudesta. Esimerkiksi MT-AA*-algoritmin kaikki toiminnallisuus on testattu testaamalla siihen liittyviä metodeja, analysointia ja tehokkuutta osissa. Myös omat tietorakenteet ovat testattuja ajallisesti siltä osin, jotka ovat olleet/voivat olla algoritmin kannalta kriittisiä.

Verkon luontia on myös testattu erikokoisilla merkkisyötteillä, joista suurin on ollut $100 * 1000$, koska tämän kokoisen verkon analysointi ja reitinhaku on sujunut alle yhdessä sekunnissa. Eniten ohjelmaa on kuitenkin testattu erilaisilla .bmp-kuvilla, jotka on analysoitu ja joissa on ajettu algoritmia. Näiden kuvien generointi on ollut helppoa ja nopeaa, joten tästä syystä ne ovat nousseet hyviksi kandidaateiksi ja vertailukohteiksi.

Apuluokat ovat testattu siltä osin, kun nähtiin tarpeelliseksi. Esimerkiksi get- ja set-metodeita ei ole testattu, koska ei ole nähty tarvetta testata näin triviaaleja metodeja.

Karttaa piirtäessä on testattu lähinnä ajallisesti, että miten kauan polun generoimisessa menee. Tämä nopeus riippuu aivan siitä, että kuinka paljon hidastetta algoritmille annetaan, jotta käyttäjällä olisi tarpeeksi aikaa klikata uusi maali algoritmille. Kuitenkin hidasteen ollessa 0 algoritmi toimii maksimaalisella teholla, joten tätä tietoa on käytetty muun muassa ohjelman suorituskyvyllisessä testauksessa.

Seuraavassa esitellään rivikattavuuden raportit. Syy miksi joissain rivikattavuus on matalahko, johtuu siitä, että joitain metodeja on käytännössä mahdoton testata JUnit-testeillä järkevästi. Näitä ovat muun muassa algoritmin keskeytykset klikkauksella sekä piirtäminen. Get- ja set-metodien testaukset on myös jätetty tarkoituksella pois testeistä.

Coverage Report - All Packages

Package	# Classes	Line Coverage	Branch Coverage	Complexity
All Packages	17	86% 604/695	77% 219/282	2,15
fi.wakr.lojikka	4	95% 91/95	82% 41/50	3
fi.wakr.lojikka.reitinhaku	4	85% 314/367	80% 104/130	2,232
fi.wakr.lojikka.tietorakenteet	7	83% 167/200	71% 70/98	1,968
fi.wakr.util	2	96% 32/33	100% 4/4	1,2

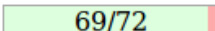
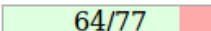
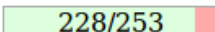
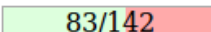
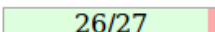
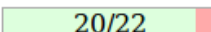
All Packages

Classes

[AStar](#) (91%)
[Analysoija](#) (96%)
[Dijkstra](#) (91%)
[Heurestiikka](#) (91%)
[Jono](#) (90%)
[Kuva](#) (96%)
[MTAA](#) (88%)
[MinimiKeko](#) (80%)
[Piste](#) (100%)
[Reitinhakija](#) (75%)
[Solmu](#) (95%)
[TaulukkoLista](#) (83%)
[Ymparistomuuttuja](#) (100%)

Number of Classes	Line Coverage	Mutation Coverage
6	92%  323/352	69%  167/241

Breakdown by Package

Name	Number of Classes	Line Coverage	Mutation Coverage
fi.wakr.logiikka	2	96%  69/72	83%  64/77
fi.wakr.logiikka.reitinhaku	3	90%  228/253	58%  83/142
fi.wakr.util	1	96%  26/27	91%  20/22

Minkälaisilla syötteillä testaus tehtiin

Testeissä on käytetty syötteenä muun muassa erikokoisia merkkikarttoja, joissa on erilaisia esteitä hidasteena. Reitinhakuun liittyvissä testeissä näiden avulla on testattu polkujen oikeellisuutta, mutta myös polkujen generoinnin nopeutta ja tästä syystä jokaisessa testissä on ollut aikarajana 1 sekunti. Toisena syöteluokkana on käytetty .bmp-kuvia, jotka on mahdollista myös olla käyttäjän itse generoimia. Näiden karttojen on oltava kuitenkin sellaisia, että maali on merkattu punaisella ja lähtö vihreällä.

Testien karttoina on ollut mm. täysin tyhjiä kenttiä, joissa maali ja lähtö ovat vastakkaisissa kulmissa, tilanteita, jossa välissä on ollut [muotoinen este sekä kartta, jossa on ollut L muotoinen este. Näistä kaikista haastavimmaksi on osoittautunut [, joka on kohtisuorassa lähtöä kohti ja maalin välissä, koska tällöin reitinhaku usein menee analysoimaan koko esteen välisen alueen, vaikka heurestiikka olisikin kunnossa. Tästä ongelmatilanteesta voitaisiin päästä eroon soveltamalla konveksi verhoa, mutta sitä ei ole toteutettu ohjelmassa sen vaativuuden vuoksi.

Polkujen lyhyimmyyden määrittelemisessä on luotu ensin tyhjä kartta, jonka lyhyin reitti on laskettu käsin. Tämän jälkeen reittiin on lisätty esteitä ja laskettu lyhyin reitti uudelleen. Algoritmin on pitänyt löytää lyhyin reitti aina esteiden lukumäärästä riippumatta.

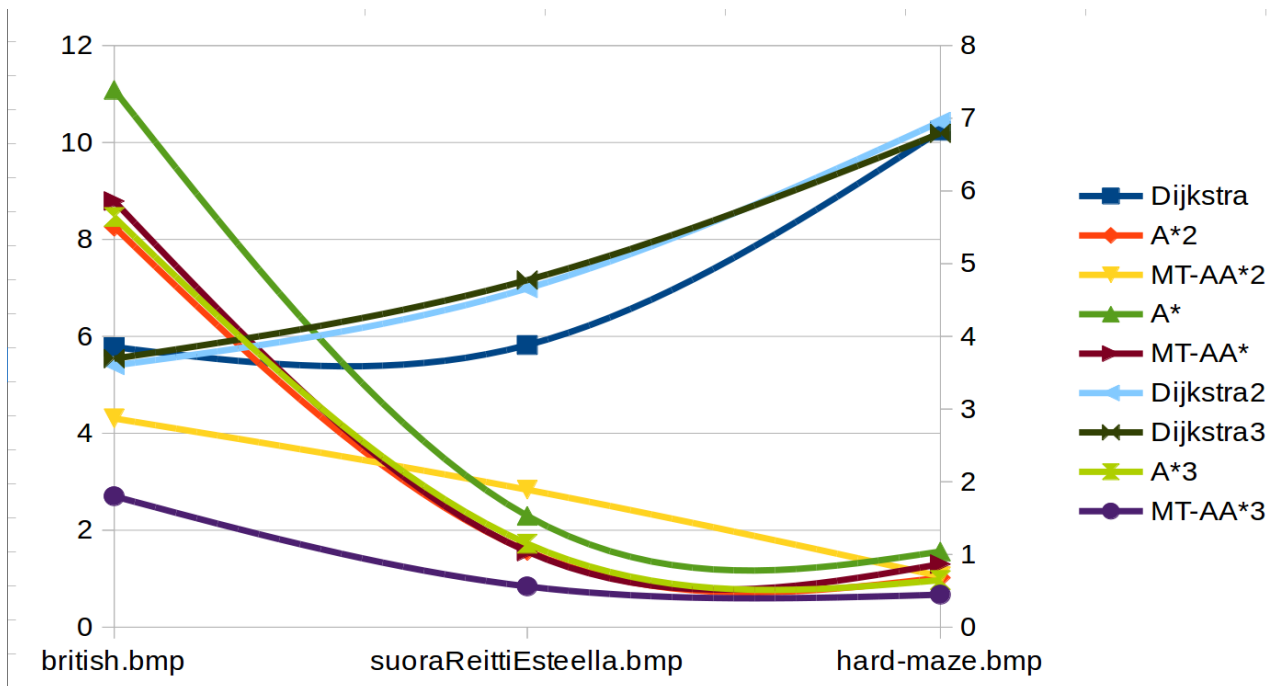
MT-AA* -algoritmiin liittyvät testit on tehty suurilla ja vaikeilla labyrinteilla, joissa navigoiminen on usealle reitinhakualgoritmillemme vaikeaa. Parhaimmat tulokset on saatu, kun on verrattu MT-AA* thesus-kartalla samalla kartalla ajettuun A*:iin sekä Dijkstraan. Näiden väliset ajat ovat saattaneet olla jopa 50% nopeammat MT-AA* hyväksi.

Miten testit voidaan toistaa

Testit voidaan toistaa luomalla samanlaiset kartat, kuin testipakkauksessa, tai luomalla omat kartat ja laskemalla ensin itse varmasti lyhyimmän reitin, jonka jälkeen algoritmi ajetaan ja tarkastetaan, että lyhyin polku on sama kuin se, mihin on itse tultu. Karttoja voidaan etsiä myös netistä ja todeta, että MT-AA* on käytännössä aina nopeampi tai vähintään yhtä nopea, kuin A* tai Dijkstra.

Muut testit voidaan toistaa käytännössä millä tahansa syötteillä, kunhan testattavat asiat ovat oikeita eli ne toimisivat myös esimerkiksi Javan omissa tietorakenteissa. Karttojen analysoinnin oikeellisuuden tarkastamisesta kannattaa tehdä ensin muutama oma kartta, joiden arvot ja mahdolliset analysoinnit ovat ylhäällä ja tarkistettu ensin itse, jonka jälkeen ohjelman analysoinnit suoritetaan ja verrataan, että tulokset vastaavat haluttua.

Tulokset graafisena



Tulokset kolmelta eri kartalta (british.bmp, suoraReittiEsteella.bmp, hard-maze.bmp), joissa ajettiin jokainen algoritmi kolme kertaa peräkkäin. Horisontaalinen akseli kuvaa kulunutta aikaa per kartta. Tuloksista huomaamme, että joskus Dijkstra on jopa nopeampi kuin A* (british.bmp), joka johtuu siitä, että jotkut kartan muodot suosivat tiettyjä algoritmeja. Tässä tapauksessa british.bmp-kartta on labyrintti, mutta kuitenkin joissain kartoissa erot ovat hyvinkin selkeät, kuten esimerkiksi kartassa hard-maze.bmp Dijkstra jää kauas muista hitaudellaan. Kuvasta nähdään myös, että jo kolmannes ajettu MT-AA* on jo aina kartasta riippumatta nopeampi kuin muut.

Hot Spots - Method	Self Time [%] ▼	Self Time	Self Time (CPU)	Total Time	Total Time (CPU)
fi.wakr.kayttoliittyma.Piirtaja.piirraKarttaanHitaasti (java.awt.Color, int)		27 928 ms (51.9%)	27 559 ms	39 971 ms	27 570 ms
fi.wakr.kayttoliittyma.Piirtaja.nuku (int)		12 039 ms (22.4%)	7,76 ms	12 039 ms	7,76 ms
fi.wakr.kayttoliittyma.Ikkuna.jMenuAvaaActionPerformed (java.awt.event.ActionEvent)		6 877 ms (12.8%)	2 151 ms	6 920 ms	2 195 ms
fi.wakr.logiikka.reitinhaku.Reitinhakija.luoVerkkoRBG ()		3 830 ms (7.1%)	3 830 ms	5 170 ms	5 170 ms
fi.wakr.logiikka.tietorakenteet.TaulukkoLista.<init> (int)		1 331 ms (2.5%)	1 331 ms	1 331 ms	1 331 ms
fi.wakr.main.Main.main (String[])		722 ms (1.3%)	722 ms	722 ms	722 ms
fi.wakr.kayttoliittyma.Ikkuna.initComponents ()		504 ms (0.9%)	499 ms	504 ms	499 ms
fi.wakr.kayttoliittyma.Ikkuna.<init> ()		192 ms (0.4%)	192 ms	696 ms	691 ms
fi.wakr.logiikka.reitinhaku.MTAA.alustaHeurestiikka ()		130 ms (0.2%)	130 ms	136 ms	136 ms
fi.wakr.logiikka.Analysioija.analysoiKarttaArvoiksiVareista (int[][], fi.wakr.logiikka.reit...		66,2 ms (0.1%)	66,2 ms	74,4 ms	74,4 ms

Thesus.bmp-kartan ajamisen jälkeen 10 pitkäkestoista funktiota MT-AA*:ssa, josta nähdään, että piirtäminen vie eniten aikaa ohjelman suorituksen aikana. A* ja Dijkstran vastaavat ajat näyttävän samanlaisilta, mutta *alustaHeurestiikka()*-funktioita ei ole.

Suorituskykytestaus

Algoritmin suorituskyvyn maksimia on testattu pääosin thesus.bmp-kartalla johtuen sen massiivisuudesta sekä seinien ja umpikujien määrästä. Tämän kartan ratkaiseminen on ollut vaikeaa mille tahansa toteutetulle reitinhaululle, jolloin se on päätyntä erinomaiseksi kohteeksi. Kuitenkin johtuen kartan massiivisuudesta on ohjelma suoritettu käyttäen komentoa `"java -jar -Xms6G MTAATiraLabra-1.0-jar-with-dependencies.jar"`. Ajojen ajat ovat jakautuneet seuraavasti 4 peräkkäiselle ajolle tällä kartalla:

	1.	2.	3.	4.
Dijkstra	26s	26s	26s	26s
A*	28s	31s	30s	23s
MT-AA*	32s	17s	10s	9s

Uuden polun etsimistä on hieman vaikea testata, joten testit ovat olleet hyvin manuaalisia. Käytännössä testit ovat kattaneet manuaalisesti, että onko polku oikein maalin vaihtuessa ja, että kauan aikaa uuden maalin etsimiseen on kulunut, jos maali vaihtuu puolella välissä hakua takaisin lähdöksi. Testissä on käytetty karttaa british.bmp-karttaa, koska se on muodoltaan paras kartta testaamaan maalin vaihtumista sen koonsa ja pohjapiirroksen vuoksi. Testissä on käytetty hidastusta arvolla 0 jokaiselle algoritmille, jotta maali ehdittäisiin vaihtaa.

	Haku A → B	Haku A → B, jossa reitti vaihtuu kesken
Dijkstra	5s	6s
A*	6s	4s
MT-AA*	8s	3s

Syy miksi tässä A* ja MT-AA*:n tulokset ovat lähellä toisiaan, johtuu siitä, että toteutettu MT-AA* ei ole puhdas MT-AA*, vaan johdannainen jossa on käytetty A*:n ideologiaa etsiä maali sen vaihtuessa.