Testausdokumentti

*Mitä on testattu, miten tämä tehtiin
*Minkälaisilla syötteillä testaus tehtiin (vertailupainotteisissa töissä tärkätä)
*Miten testit voidaan toistaa
*Ohjelman toiminnan empiirisen testauksen tulosten esittäminen graafisessa muodossa.
*Testaus on ideaalitapauksessa suoritettava ohjelma. Tällöin testi on helposti toistettavissa, mikä helpottaa toteutuksen tekoa jo varhaisessa vaiheessa. On erittäin suositeltavaa käyttää testaukseen JUnitia.

Testit

* Itse toteutettujen tietorakenteiden (tira-paketti) kaikki julkiset metodit on yksikkötestattu, lukuunottamatta Pari-luokkaa.

* AStar-haulla on muutama testi (haku-paketti): haku ylipäänsä toimii ja eri laskimet reiteille toimivat.

* App-luokassa (käynnistysluokka) on kattavasti debug-metodeja, joilla olen toteutusvaiheessa varmistanut luokkien toimivuutta.

* Useissa luokissa on lisäksi mukana debug-metodeja, jotka keräävät ja tulostavat/palauttavat tietoa luokkien toiminnasta.

* App-luokan testikäyttöliittymällä voi testata reitinhakua sekä pysäkkiverkossa että satunnaisgeneroidussa verkossa. Reittihaku suoritetaan aina kymmenen kertaa, ja suorituksen jälkeen tulostetaan saatu reitti ja keskiarvo kuluneesta ajasta, pienin aika sekä suurin aika.

* Gui-luokan testikäyttöliittymässä voi luoda uuden satunnaisen verkon sekä suorittaa hakuja pysäkki- ja satunnaisessa verkossa. Haun tulokset piirretään.

Testien tuloksia: suorituskykytestausta

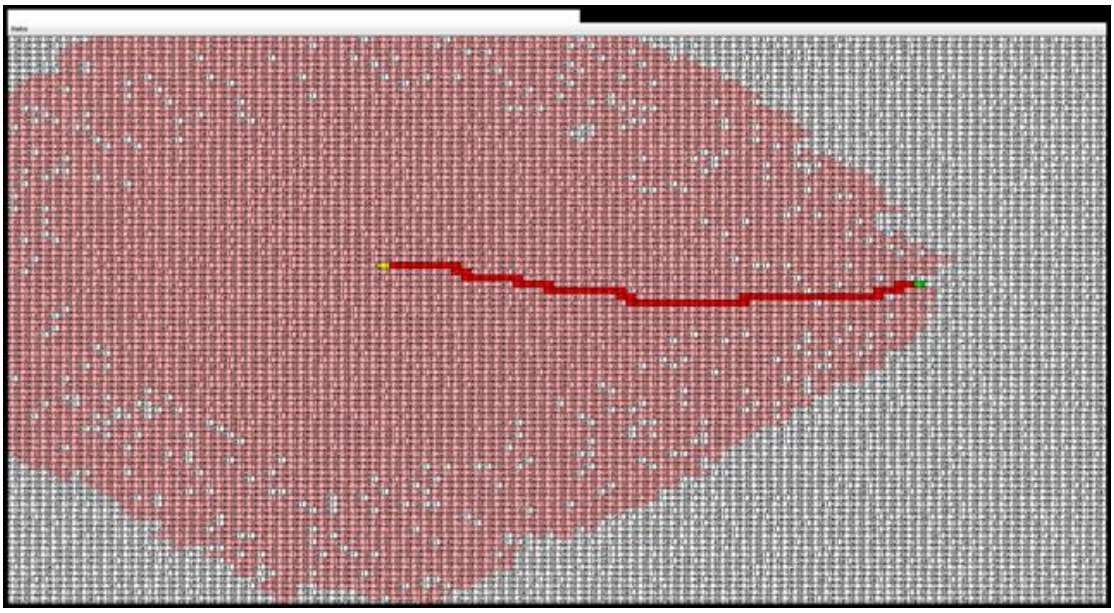
Kuvissa kuljettu reitti on punaisella, käydyt solmut vaaleanpunaisella, alkusolmu keltaisella ja loppusolmu vihreällä.

1. AStar-haku ja heuristiikka

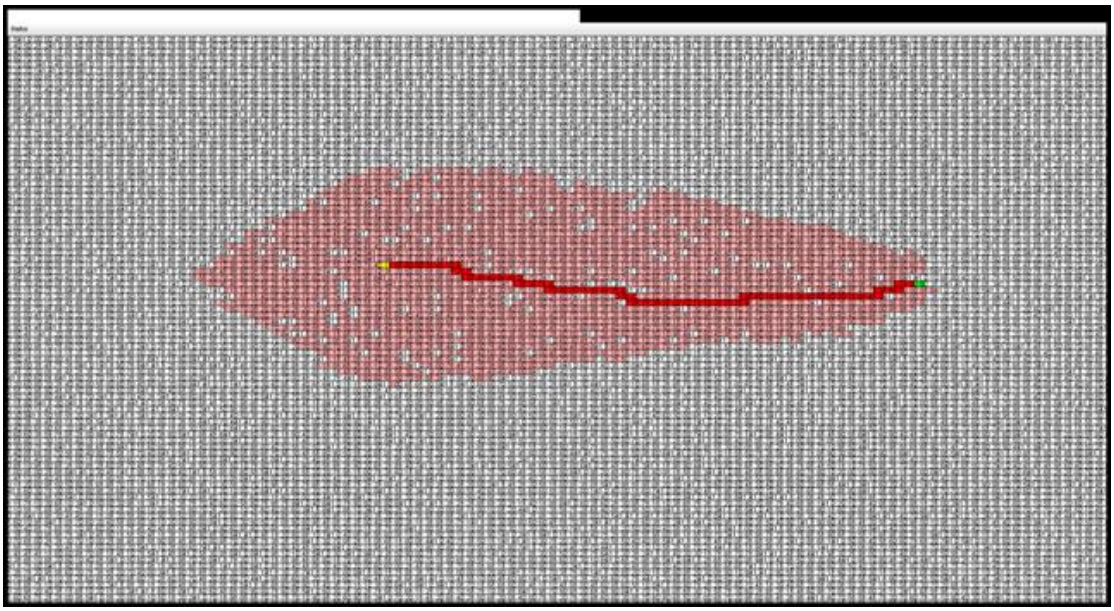
Toistaminen:
Asetetaan aluksi käyttöön uusi satunnainen verkko syötteellä 300:300:1:2:2:1. Valitaan sitten laskimeksi BFS, suoritetaan haku ja sitten valitaan heuristinen laskin ja suoritetaan haku uudestaan.

Solmujen käsittelyjärjestyksen määrää niiden hyvyys g(n): reitin kustannus f(alku,n) ja arvioitu jäljellä oleva kustannus h(n).

Heuristisen funktion h(n) arvo vaikuttaa haun toimintaan. Jos h(n)=0, saadaan leveyssuuntainen haku:



Sama haku suoritettuna siten, että h(n)=|n.x-maali.x|+|n.y-maali.y|:



Solmuja ei ole siis käsitelty niin paljoa turhaan.

Mitä paremmin h(n) vastaa oikeaa jäljellä olevaa kustannusta f(n,maali), sitä tehokkaammin haku toimii. Tätä voi kokeilla valitsemalla verkon siten, että minimiPaino on hyvin suuri.

Jos heuristinen arvio yliarvioi jäljellä olevaa kustannusta, ei AStar-haku anna parasta mahdollista ratkaisua. Omassa toteutuksessani koko haku saattaa kaatua tällöin (johtuu käsittelyjärjestyksen toteutustavasta: nykyisellään vaatimuksena on, että g(n) ovat kasvavia käsittelyjärjestyksessä edetessä).

Tämän voi toistaa valitsemalla minimiPainon, joka on arvoltaan alle yksi.

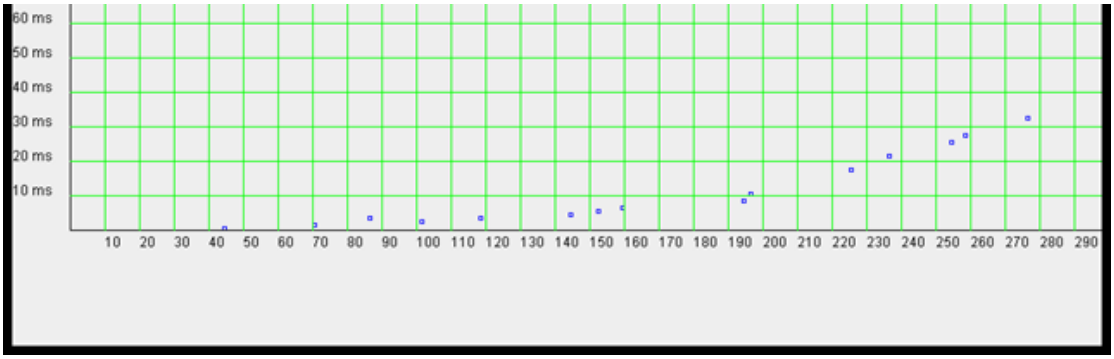
2. AStar-haku ja verkon tiheys

Toistaminen: syötteellä 300:300:1:2:2:1 luotu verkko ja syötteellä 300:300:1:2:2:0 luotu verkko. Suoritetaan näillä testejä-painikkeen mukaiset testit.

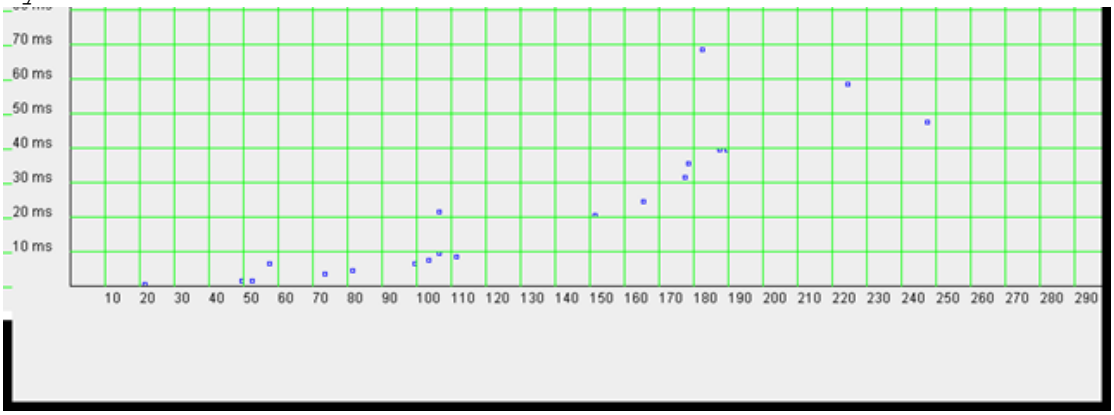
Oletusarvona verkossa voi kulkea vain koordinaattiakselien suuntaisesti. Valinnalla 300:300:1:2:2:0 luotu verkko sallii liikkumisen myös sivuittain.

Minimiaikavaatimus verkossa kulkemiseen on O(k^d), jossa k on naapureiden lukumäärä kullekkin solmulle ja d on reitin pituus. Sivuttaisen liikkumisen sallivassa verkossa naapureita on 8, vain akselien suuntaisen liikkumisen sallivassa verkossa 4. Annetulla reitin pituudella d minimiaikavaatimuksien suhde on siis 2^d. Jos sivuttaisuuntainen kulkeminen sallitaan, lyhyin mahdollinen reitti on puolet siitä jos kulkeminen estetään. Kuitenkin suoritusajassa näkyy selvä ero.

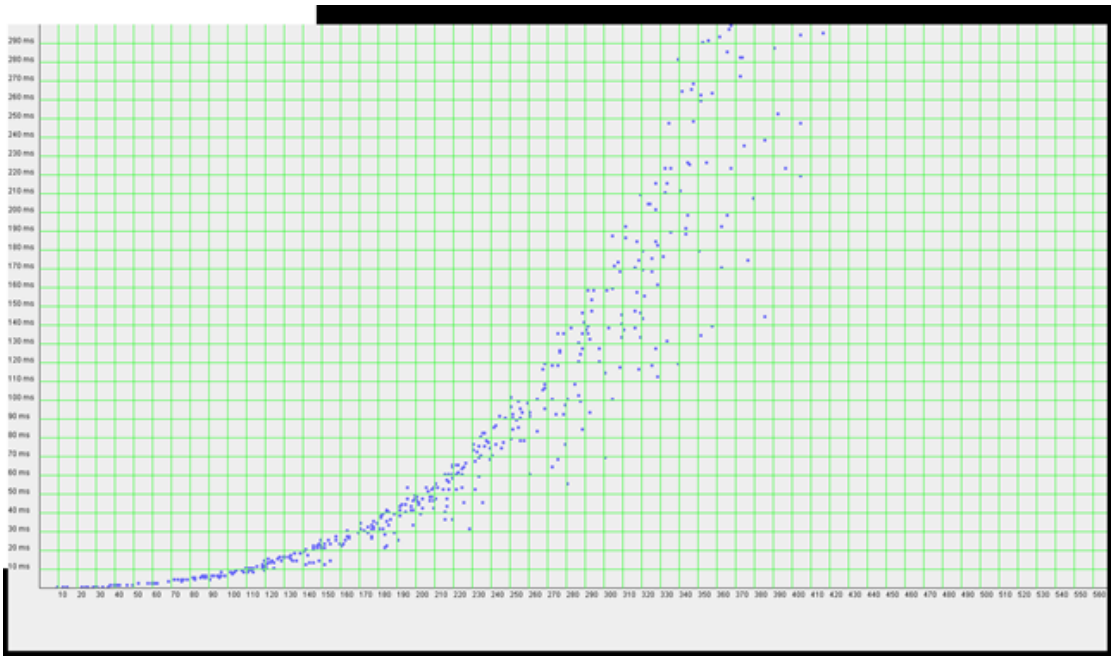
Vain akselien suuntaan:



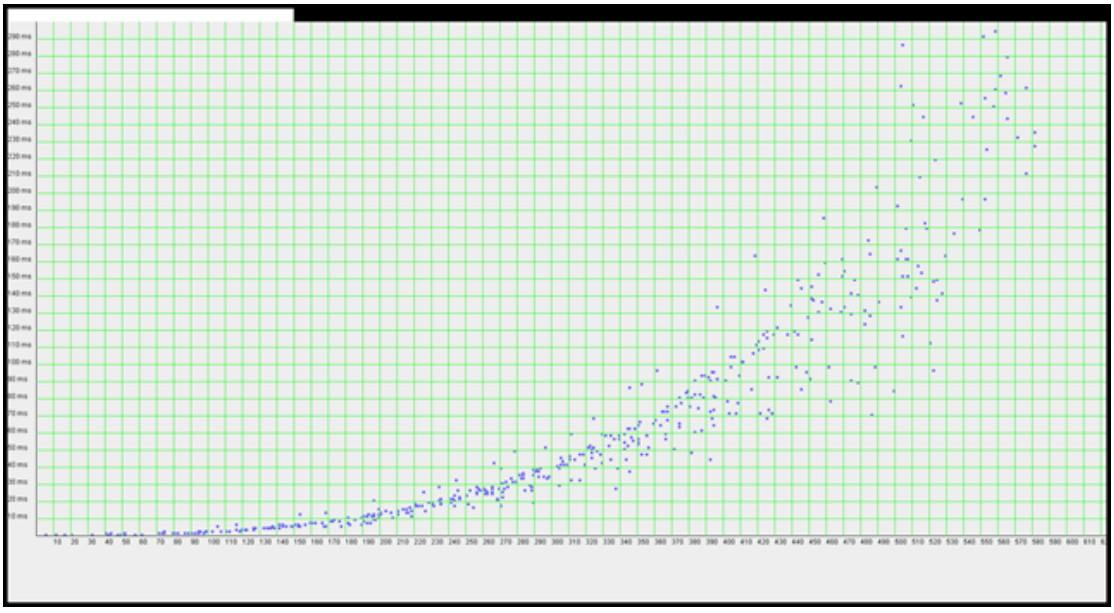
Myös sivusuunnassa:



500x500 verkossa 500 näytteellä ero korostuu entisestään. Vain akselien suuntaan:



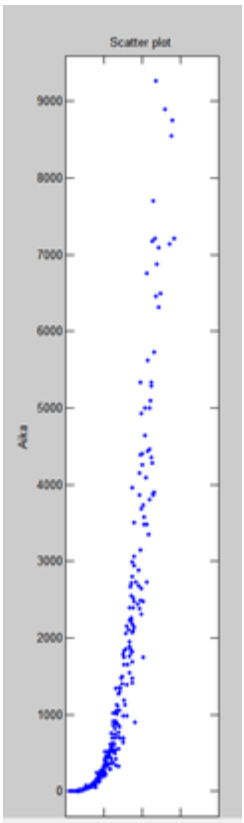
Myös sivusuunnassa:



3. Verkon ja polun koko

Toistaminen: luodaan syötteellä 1000:1000:1:2:2:1 verkko, jonka koko on 1000x1000. Nyt ohjelman piirtämään kuvaajaan ei mahdu kaikki pisteet. Myös kokonaissuoritusaika on melkoinen.

Suurilla verkoilla ja siten myös pitkillä poluilla aikavaatimus kasvaa eksponentiaalisesti:


Scatter plot
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
Aka