TEKNILLINEN KORKEAKOULU
Laskennallisen tieteen ja tekniikan erikoistyö
20.9.2007

O(N) nopea marssimenetelmä

Veikko Sariola
etunimi.sukunimi@tkk.fi
57747H

TEKNILLINEN KORKEAKOULU ERIKOISTYÖN TIIVISTELMÄ

Laskennallisen tieteen ja tekniikan erikoistyö

Tekijä

Veikko Sariola

Päiväys

20.9.2007

Opintokirja

57747H

Työn nimi

O(N) nopea marssimenetelmä

Työn tarkastaja

Prof. Timo Eirola

Nopeat marssimenetelmät ovat vakiintuneet eikonaaliyhtälön numeeriseksi ratkaisemiseksi. Näiden menetelmien laskennallinen vaativuus on $O(N \log N)$, missä N on diskretointiverkon solmujen lukumäärä. Tässä erikoistyössä esitellään hiljattain julkaistu nopea marssimenetelmä, jonka vaativuus on vain $O(N)$, mutta virhe samaa suuruusluokkaa kuin alkuperäisen menetelmän. Kokeellisessa osassa uuden ja vanhan menetelmän tarkkuutta verrataan yksinkertaisissa tehtävissä, jotka voidaan ratkaista myös analyyttisesti. Lopuksi uutta menetelmää sovelletaan laskettaessa pistemäisen valonlähteen varjoja sekä etsittäessä robotin reittiä. Uusi menetelmä osoittautuu käytännön tehtävissä huomattavasti vanhaa menetelmää nopeammaksi ja virheen muutos on merkityksetön.

Avainsanat

Nopeat marssimenetelmät, Eikonaaliyhtälö, Epäsiisti prioriteettijono

Sisällysluettelo

1 Johdanto

Eikonaaliyhtälö on epälineearinen osittaisdifferentiaaliyhtälö, joka esiintyy monissa ongelmissa; mm. plastisen aineen mallintamisessa (Alouges ja Desimone 1999), kuvan segmentoinnissa (Sofou ja Maragos 2003), laskettaessa valonlähteen aiheuttamia varjoja (Sethian 1999, s. 49) ja etsittäessä lyhintä reittiä robotille (Garrido et al. 2006). Tänä päivänä kuitenkin tunnetuin käyttökohde lienee **tasa-arvomenetelmien** (engl. level set methods) yhteydessä, koska eikonaaliyhtälö esiintyy mm. ratkaistaessa **etumerkillä varustettuja etäisyysfunktioita** (engl. signed distance functions). Eikonaaliyhtälön analyyttinen ratkaiseminen onnistuu vain erittäin yksinkertaisissa tapauksissa. Yleensä yhtälö voidaan ratkoa vain numeerisesti.

Nopeat marssimenetelmät ovat vakiintuneet eikonaaliyhtälön numeeriseksi ratkaisemiseksi. Yleensä nopeissa marssimenetelmissä eikonaaliyhtälö diskretoidaan karteesisessa koordinaatistossa. Niitä kutsutaan "nopeiksi", koska algoritmi pyyhkäisee vain kerran koko verkon yli, rakentaen ratkaisua edetessään. Näiden menetelmien laskennallinen vaativuus on $O(N \log N)$, missä N on diskretointiverkon solmujen lukumäärä.

Tässä erikoistyössä esitellään hiljattain julkaistu nopea marssimenetelmä, jonka vaativuus on vain $O(N)$, mutta virhe samaa suuruusluokkaa kuin alkuperäisen menetelmän. Uusi menetelmä osoittautuu käytännön tehtävissä huomattavasti vanhaa menetelmää nopeammaksi ja virheen muutos on merkityksetön.

Tässä työssä käsitellään vain kaksiulotteista eikonaaliyhtälöä; kaikki tekniikat ovat kuitenkin laajennettavissa useampiin ulottuvuuksiin.

1.1 Eikonaaliyhtälö

Eikonaaliyhtälö on epälineaarinen osittaisdifferentiaaliyhtälö muotoa

$$\begin{aligned} &|\nabla u(x)| = f(x), \ x \in \Omega \\ &f(x) > 0 \ \forall \ x \in \Omega \\ &u(x) = g(x), \ x \in \Gamma \subset \Omega \end{aligned} \tag{1}$$

Tehtävässä f, g, alue Ω ja joukko Γ ovat annettuina ja ratkaistavaksi jää funktio u. Yksi tulkinta eikonaaliyhtälölle on, että u on kasvavan aaltorintaman saapumisaika kuhunkin pisteeseen ja f kuvaa hitautta kussakin pisteessä. Erikoistapaus $f = 1$, $g = 0$ antaa ratkaisuksi lyhimmän etäisyyden kustakin pisteestä Γ:aan. Positiivisuudesta johtuen eikonaaliyhtälö voidaan myös esittää muodossa

$$|\nabla u(x)|^2 = f^2(x) \tag{2}$$

Eikonaaliyhtälö on erikoistapaus Hamilton-Jacobi yhtälöistä, jotka yleisesti ovat muotoa

$$H(\nabla u(x), x) = 0 \tag{3}$$

Eikonaaliyhtälölle $H(\nabla u(x), x) = |\nabla u(x)| - F(x)$.

1.2 Nopeat marssimenetelmät

Sethian (1996) esitteli **nopeat marssimenetelmät** (engl. fast marching methods) eikonaaliyhtälön numeeriseksi ratkaisemiseksi. Sethianin esittelemässä muodossaan yhtälö 2 diskretoidaan karteesisessa koordinaatistossa muotoon

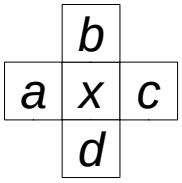
$$max(max(D_{ij}^{-x}u,0),-min(D_{ij}^{+x}u,0))^2+ \\ max(max(D_{ij}^{-y}u,0),-min(D_{ij}^{+y}u,0))^2=f_{ij}^2 \quad (4)$$

missä D_{ij}^{+x} tarkoittaa eteenpäin derivointia vaakasuunnassa, $D_{ij}^{+x}u=(u_{i+1,j}-u_{i,j})/h$, h on verkon diskretointiväli. D_{ij}^{-x} tarkoittaa taaksepäin derivointia vaakasuunnassa, $D_{ij}^{-x}u=(u_{i,j}-u_{i-1,j})/h$. D_{ij}^{+y} ja D_{ij}^{-y} vastaavasti pystysuunnassa.

Käytännössä yhtälöä 4 käytetään u:n arvon ratkaisemiseksi solmussa (i,j). Solmun arvo riippuu sen naapurisolmujen (vaaka- ja pystysuunnassa viereisten) solmujen arvoista.

Yhtälön 4 vasen puoli on oleellisesti toisen asteen yhtälö; ratkaisuista valitaan aina suurempi nk. viskositeettiratkaisun saamiseksi (Sethian 1996). Käytännössä laskeminen tapahtuu seuraavanlaisesti:

Oletetaan, että naapurisolmujen arvot ovat a, b, c ja d ja lasketaan solmun arvoa x (kuva 1).



Kuva 1: Lasketaan solmun arvo x sen naapureiden arvoista a, b, c ja d

Yleisyyttä rajoittamatta voidaan olettaa, että $h=1$ ($1/h$ voidaan ottaa yhteiseksi tekijäksi ja sisällyttää tekijään f_{ij}^2). Silloin yhtälö 4 saa muodon (lyhennetään $f=f_{ij}$)

$$max(max((x-a),0),-min((c-x),0))^2+ \\ max(max((x-b),0),-min((d-x),0))^2=f^2 \quad (5)$$

Toisaalta, tämä voidaan kirjoittaa

$$max(max((x-a),0),max((x-c),0))^2+ \\ max(max((x-b),0),max((x-d),0))^2=f^2 \quad (6)$$

Yleisyyttä rajoittamatta voidaan olettaa, että $a\le c$ ja $b\le d$. Silloin yhtälö 6 saa muodon

$$max((x-a),0)^2+max((x-b),0)^2=f^2 \quad (7)$$

Edelleen, yleisyyttä rajoittamatta voidaan olettaa, että $a\le b$.

Jos nyt $a+f\le b$, niin yhtälön 7 ratkaisu on $x=a+f$, joka on helppo tarkistaa sijoittamalla. Jos taas $a+f>b$ niin yhtälön $(x-a)^2+(x-b)^2=f^2$ ratkaisut ovat yhtälön 7 ratkaisuja. Tästä tulee toiseen asteen yhtälö $2x^2+(-2a-2b)x+a^2+b^2-f^2=0$, jonka diskriminantti $D=-4a^2+8ba-4b^2+8f^2$. Aikaisempien oletusten perusteella voidaan näyttää, että $D\ge 0$ eli yhtälölle saadaan reaalisia ratkaisuja. Yhtälön ratkaisut ovat $x=1/2(a+b)\pm 1/4\sqrt{D}$; näistä valitaan suurempi, kuten aiemmin todettiin.

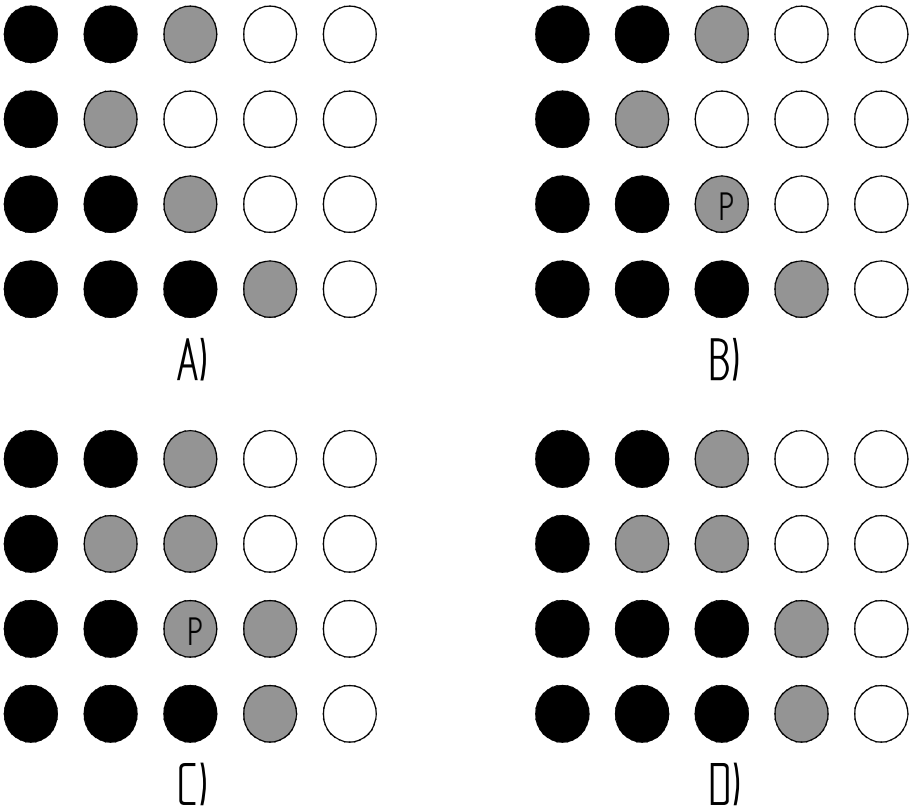
Kaikissa tapauksissa pätee, että $x\ge a$ (oletusten perusteella a oli pienin x:n naapureista) ja x:n arvo riippuu vain sen pienemmästä naapurista vaaka- ja pystysuunnassa. Tämän takia koko yhtälön ratkaisua voidaan lähteä rakentamaan pienimmistä arvoista kohti suurempia. Tarkan todistuksen tälle esitti Sethian (1996, s. 9).

Käytännössä algoritmi etenee seuraavanlaisesti. Kullakin hetkellä pidetään muistissa, mitkä

solmuista on jo *hyväksytty*, mitkä *lähellä* ja mitkä *kaukana*. Aluksi laitetaan Γ :aa vastaavat verkon solmut joukkoon *hyväksytty* ja lisätään näiden naapurit joukkoon *lähellä*. Lasketaan kaikkien *lähellä* joukon solmujen u käyttäen yhtälöä 4. Kaikki loput pisteet laitetaan joukkoon *kaukana*. Toistetaan seuraavaa:

1. Etsitään joukosta *lähellä* pienimmän u -arvon omaava solmu nimeltään *p.*
2. Laitetaan *p:*n kaikki naapurit, jotka eivät ole jo joukossa *hyväksytty*, joukkoon *lähellä*. Eli jos naapuri on joukossa *kaukana*, siirretään se joukkoon *lähellä*.
3. Lasketaan kaikkien *p*:n naapureiden, jotka ovat joukossa *lähellä*, arvot uudestaan käyttäen yhtälöä 4.
4. Siirretään *p* joukosta *lähellä* joukkoon *hyväksytty.*
5. Siirrytään kohtaan 1, jos joukko *lähellä* ei ole tyhjä.

Kuvassa 2 on havainnollistettu algoritmin etenemistä. Menetelmä on läheistä sukua kapeakaistatasa-arvomenetelmille (engl. narrow band level set methods).



Kuva 2: Nopeiden marssimenetelmien eteneminen. Mustat solmut kuuluvat joukkoon "hyväksytty", harmaat joukkoon "lähellä" ja valkoiset joukkoon "kaukana". A) Alkutilanne. B) Etsitään pienin solmu p joukosta "lähellä". C) Siirretään p:n naapurit joukkoon lähellä ja lasketaan niiden arvot. D) Siirretään p joukkoon "hyväksytty" ja palataan silmukan alkuun.

Nopeiden marssimenetelmien perusimplementaatiolla on laskennallinen vaativuus $O(N \log N)$, missä N on verkon solmujen lukumäärä. Silmukassa kaikki muut vaiheet ovat laskennalliselta vaativuudeltaan $O(1)$, paitsi 1. vaihe. Pienimmän alkion etsiminen joukosta *lähellä* onnistuu tehokkaimmiten pitämällä *lähellä* joukkoa **prioriteettijonossa** (engl. priority queue).

Prioriteettijono toteutetaan käytännössä **keko** (engl. heap) tietorakenteella, jolloin pienimmän alkion etsimisen laskennallinen vaativuus on $O(\log M)$, missä M on alkioiden lukumäärä joukossa *lähellä*. Koska aina $M \leq N$, niin pienimmän alkion etsiminen on myös $O(\log N)$. Silmukka käy vain kerran jokaisessa solmussa, joten koko algoritmille saadaan laskennalliseksi vaativuudeksi $O(N \log N)$.

Jatkossa tähän menetelmään viitataan **vanhana** menetelmänä.

Vaikka naapurisolmujen arvot tunnettaisiin tarkasti (ts. naapurisolmut kuuluisivat joukkoon Γ), gradientin ensimmäisen kertaluvun approksimaatiosta seuraa, että solmun arvoa laskettaessa tulee virhe $O(h^2)$. Näytetään tämä yksiulotteisessa tehtävässä

$$|u'(x)| = f > 0 \tag{8}$$

Aproksimoidaan derivaattaa taaksepäin ja merkataan virheen suuruusluokka

$$|(u_i - u_{u-1})/h + O(h)| = f_i \tag{9}$$

Tämä on sama asia kuin $(u_i - u_{u-1})/h + O(h) = \pm f_i$. Ratkaistaan u_i ja valitaan ratkaisuista suurempi, jolloin saadaan

$$u_i = u_{i-1} + f_i h + O(h^2) \tag{10}$$

Aproksimoitaessa derivaattaa eteenpäin saadaan vastaavanlainen tulos. Siten myös approksimoitaessa ylöspäin (eli valitaan taaksepäin tai eteenpäin derivointi riippuen kumpi on suurempi), saadaan vastaavanlainen tulos. Todistus on idealtaan samankaltainen useammissa ulottuvuuksissa.

1.3 O(N) nopea marssimenetelmä

Yatziv et al. (2006) esittivät tietääkseni ensimmäisenä seuraavanlaisen $O(N)$ nopean marssimenetelmän. Heidän tekniikkansa perustui kahteen yksinkertaiseen havaintoon:

1. Mikäli *lähellä* keon pienimmän alkion $u = u_{min}$ niin keon kaikkien muiden alkioiden $u < u_{min} + I_{max}$, missä I_{max} on funktiosta f ja diskretointivälistä h riippuva vakio.
2. Menetelmä antaa lähes yhtä hyviä tuloksia, vaikka prioriteettijonosta ei valittaisikaan aina absoluuttisesti pienintä alkiota, vaan riittävän pieni alkio.

Tarkalleen ottaen menetelmässä korvataan prioriteettijono **epäsiistillä prioriteettijonolla** (engl. untidy priority queue), jossa väli $[u_{min}, u_{min} + I_{max}]$ jaetaan tasaisesti d:hen osaväliin ts. osaväli $k \in 1..\mathrm{d}$ on $[u_{min} + (k-1) I_{max}/d \, , u_{min} + k I_{max}/d]$. Jokaista osaväliä vastaa muistissa linkitetty lista, jossa säilytetään kyseiselle osavälille osuvia alkioita. Kutsutaan näitä listoja *säiliöiksi*. Näitä säiliöitä on d kappaletta, joten niitä voidaan pitää taulukossa.

Sen sijaan, että poistettaisiin pienin alkio, poistetaankin ensimmäisen osavälin ensimmäinen alkio. Olkoon poistetun alkion u-arvo u_p. Pätee, että $u_p < u_{min} + I_{max}/d$.

Linkitetyn listan lisäys- ja poisto-operaatiot ovat laskennalliselta vaativuudeltaan $O(1)$. Alkion arvoa vastaavan säiliön etsiminen on vaativuudeltaan $O(1)$, koska osavälien tasavälisyydestä johtuen oikean säiliön etsiminen vaatii vain yksinkertaista aritmetiikkaa.

Säiliön tyhjentyessä siirrytään poistamaan alkioita seuraavasta säiliöstä. Jos kuitenkin $d \ll M$, missä M on solmujen määrä joukossa *lähellä*, tähän kuluva aika on merkityksetön. Näin koko algoritmin laskennalliseksi vaativuudeksi tulee $O(N)$.

Valitsemalla $d \sim 1/h$, voidaan osoittaa, että laskettaessa solmun arvoa väärästä

valitsemisjärjestyksestä johtuva lisävirhe on $O(h^2)$. Tämän todistamisen idea perustuu siihen, että poistetun alkion $\varepsilon=(u_p-u_{min})\leq I_{max}/d$ ja $I_{max}=O(h)$. Siten $\varepsilon=O(h^2)$, eli lisävirhe on samaa suuruusluokkaa kuin yhtälön diskretoinnista johtuva virhe ja kokonaisvirhe samaa suuruusluokkaa kuin vanhan menetelmän. Tarkan todistuksen tälle esittivät Yatziv et al. (2006, s. 4).

Siten saavutetaan $O(N)$ nopea marssimenetelmä, jonka virhe on samaa suuruusluokkaa kuin vanhan menetelmän.

On syytä mainita, että poiketen alkuperäisestä menetelmästä uudessa menetelmässä on merkittävää, missä järjestyksessä naapurialkiot lisätään epäsiistiin prioriteettijonoon, koska kustakin säiliöstä poistetaan ensimmäisenä alkio, joka laitettiin sinne ensimmäisenä. Käytännön testit kuitenkin osoittavat, että väärästä poistojärjestyksestä johtuva virhe on huomattavasti teoreettista maksimia pienempi.

Jatkossa tähän menetelmään viitataan **uutena** marssimenetelmänä.

2 Numeerisia kokeita

Seuraavia kokeita varten sekä vanha että uusi menetelmä implementoitiin käyttäen C# ohjelmointikieltä Microsoftin .NET alustalla. Algoritmin tarvitsema alkudata (funktiot f ja g) generoitiin Matlabilla ja tallennettiin tiedostoihin, joista ohjelma latasi datansa. Aikamittauksissa ei ole huomioitu datan tallentamiseen tai lataamiseen tarvittavaa aikaa; datan lataaminen on $O(N)$ operaatio ja joudutaan tekemään yhtälailla algoritmin molemmissa versioissa, joten sen huomiotta jättäminen on perusteltua. Lisäksi monissa käytännön tilanteissa saatetaan data ladata vain kerran ja ajaa algoritmia (lähes) samalla datalla useasti; esimerkiksi reitinetsintätehtävissä kartta saattaa pysyä samana, mutta määränpää vaihtua eri ajokertojen välissä.

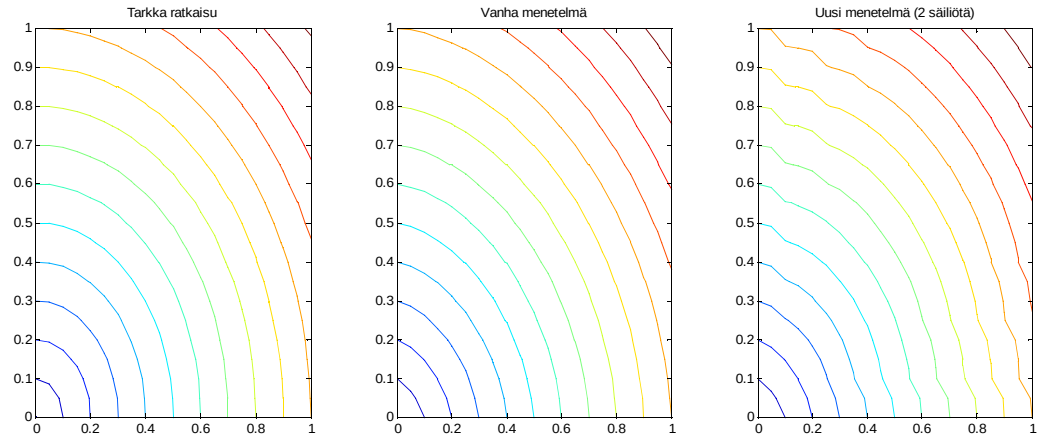
Molemmissa implementaatioissa säilytettiin muistissa kullekin verkon solmulle solmun sijainti (epäsiistissä) prioriteettijonossa, ja tätä päivitettiin solmun prioriteetin (u:n) mahdollisesti muuttuessa. Arvon tallentaminen muistiin on $O(1)$, joten se ei lisännyt laskennallista vaativuutta. Näin solmun prioriteetin vaihtuessa ei tarvitse käydä läpi koko jonoa, vaan alkion hakeminen jonosta on myös $O(1)$ operaatio.

2.1 Etäisyys origoon

Ensimmäisen kokeen tehtävänä on osoittaa, että virhe saadaan uudella menetelmällä pysymään karkeasti ottaen samansuuruisena kuin diskretoinnin aiheuttama virhe. Kokeillaan algoritmia tehtävälle:

$$\begin{aligned} &|\nabla u(x)|=1,\ x\in[0,1]x[0,1] \\ &u(x)=0,\ x\in\{(0,0)\} \end{aligned} \qquad (11)$$

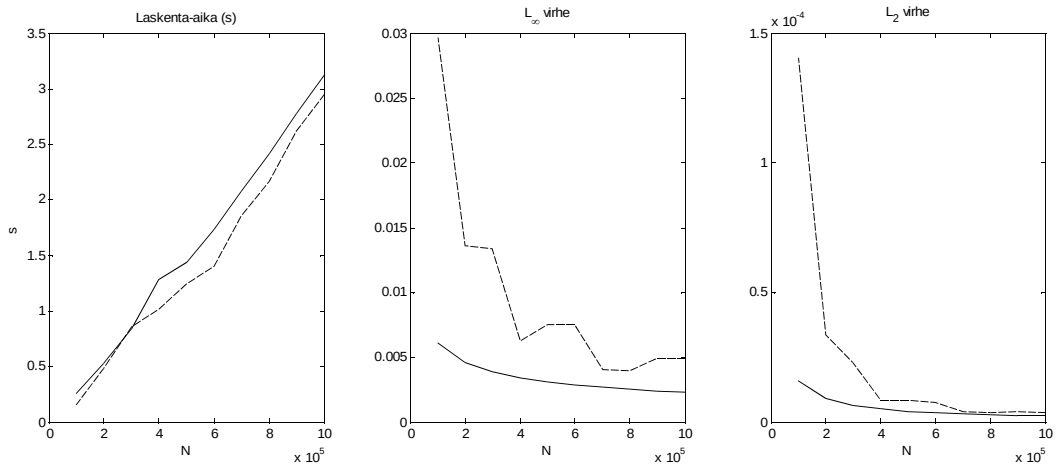
Tarkka ratkaisu tälle on etäisyys origoon alueessa $[0,1]x[0,1]$. Kuvassa 3 on esitetty $21\,x\,21$ verkolla tarkka, vanhan menetelmän ja uuden menetelmän antama ratkaisu.



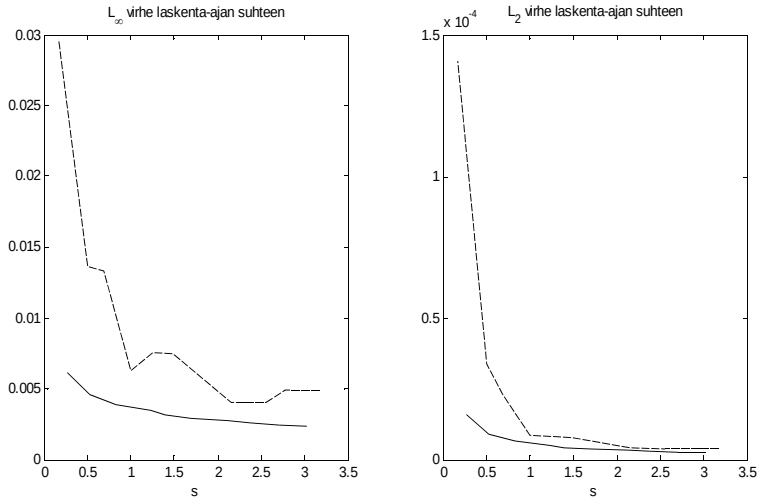
Kuva 3: Etäisyys origoon yksikköneliössä. Vasemmalta oikealle: Tarkka, vanhan menetelmän ja uuden menetelmän ratkaisu

Kokeillaan sitten muuttaa verkon kokoa, valiten aina $d \sim 1/h$. Tässä on valittu $d = ceil(0.02/h)$, missä *ceil* tarkoittaa pyöristämistä ylöspäin. Kuvassa 4 on esitetty laskenta-aika, L_∞ ja L_2 virheet verkon solmujen lukumäärän N suhteen sekä vanhalle että uudelle menetelmälle. Nähdään, että laskenta-aika on pienempi kuin alkuperäisellä versiolla, mutta L_∞ virhe on suuruusluokaltaan sama kuin alkuperäisen menetelmän virhe.

Kuvassa 5 on esitetty virheet laskenta-ajan suhteen. Vaikka tähän mennessä vanha menetelmä näyttää olevan parempi kuin uusi, on syytä huomauttaa, että tehtävän symmetrisyys johtaa alkioiden epätasaiseen jakautumiseen säiliöihin, jolloin pitää käyttää suurta määrää säiliöitä tarkkuuden saamiseksi. Myöhemmissä esimerkeissä huomataan, ettei tarvita läheskään yhtä paljon säiliöitä, jos jakautuminen on tasaisempaa. Koe kuitenkin vahvistaa, että molemmissa menetelmissä verkon solmuja lisäämällä virhe pienenee ja näyttäisi lähestyvän nollaa.



Kuva 4: Laskenta-aika, L_∞ ja L_2 virheet verkon koon N funktiona laskettaessa etäisyyttä origoon alueessa $[0,1]x[0,1]$. Yhtenäinen viiva: vanha menetelmä. Katkoviiva: uusi menetelmä.



Kuva 5: L_∞ *ja* L_2 *virheet laskenta-ajan funktiona laskettaessa etäisyyttä origoon alueessa* $[0,1]x[0,1]$ *. Yhtenäinen viiva: vanha menetelmä. Katkoviiva: uusi menetelmä.*

2.2 Etäisyys kahteen pisteeseen

Kokeillaan seuraavaksi ratkoa vain aavistuksen verran monimutkaisempia tehtäviä

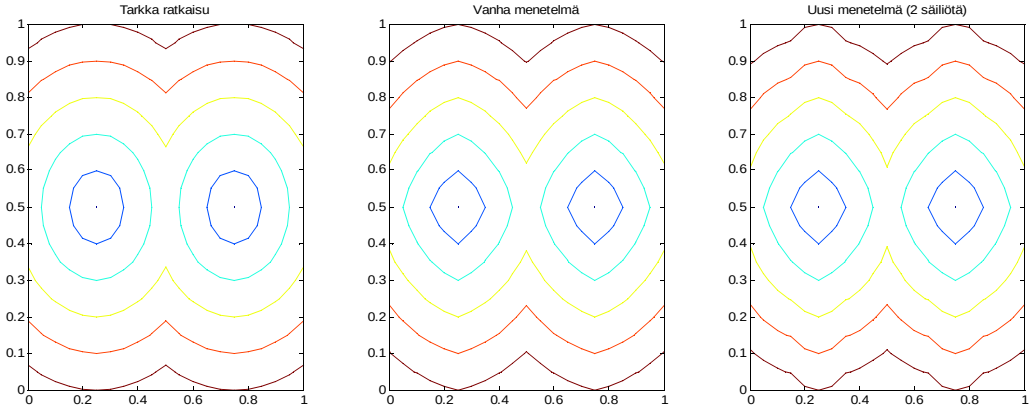
a)
$$\begin{aligned} &|\nabla u(x)| = 1\,, \quad x \in [0,1]x[0,1] \\ &u(x) = 0\,, \quad x \in \{(0.25,0.5),(0.75,5)\} \end{aligned} \tag{12}$$

b)
$$\begin{aligned} &|\nabla u(x)| = 1\,, \quad x \in [0,1]x[0,1] \\ &u(x) = 0\,, \quad x \in \{(0.25,0.25),(0.75,75)\} \end{aligned} \tag{13}$$

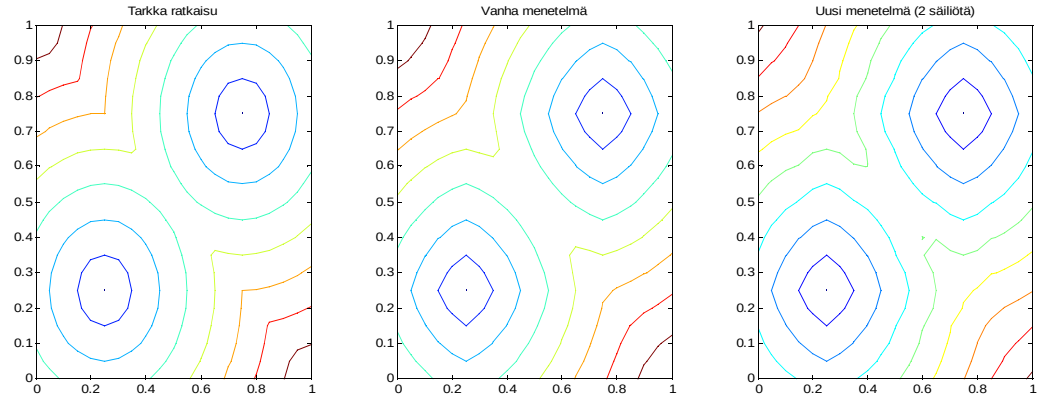
Tarkka ratkaisu tehtäville on kussakin pisteessä lyhin etäisyys pisteisiin a) (0.25,0.5) ja (0.75,0.5) b) (0.25,0.25) ja (0.75,0.75).

Menetelmissä käytettävät parametrit valitaan kuten edellisessä kokeessa. Kuvassa 6 on esitetty $21\,x\,21$ verkolla a)-kohdan tarkka, vanhan menetelmän ja uuden menetelmän antama ratkaisu ja kuvassa 7 b)-kohdan. Kuvassa 8 a)-kohdan laskenta-ajat ja virheet eri menetelmille ja kuvassa 9 b)-kohdan.

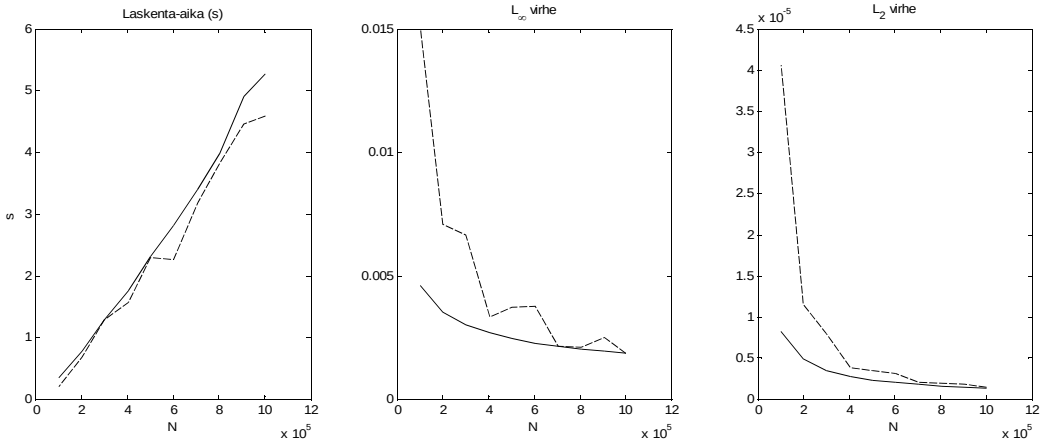
Kuvassa 10 on esitetty virheet laskenta-ajan suhteen a)-tehtävälle ja kuvassa 11 b)-tehtävälle. Näissä tehtävissä uusi menetelmä on jo joissakin tapauksissa nopeampi kuin vanha menetelmä.



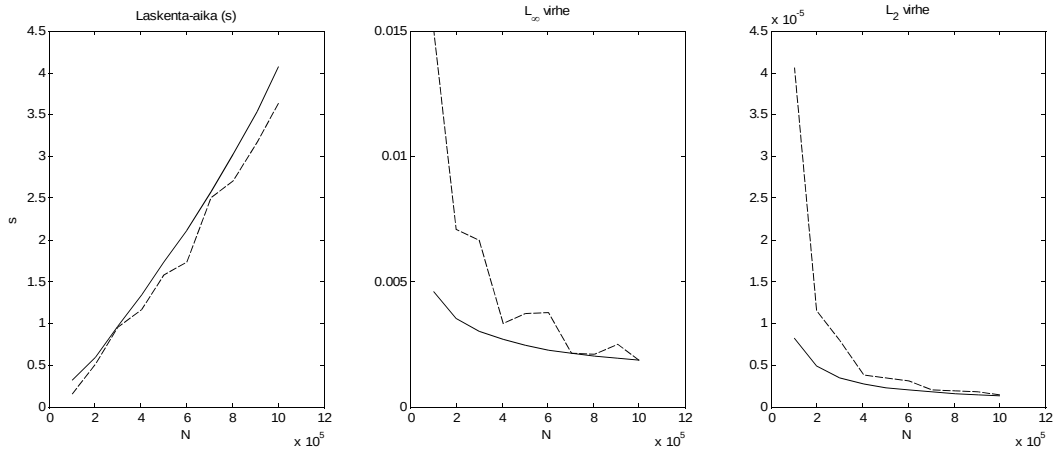
Kuva 6: Pienin etäisyys pisteisiin (0.25,0.5) ja (0.75,0.5) yksikköneliössä. Vasemmalta oikealle: Tarkka, vanhan menetelmän ja uuden menetelmän ratkaisu.



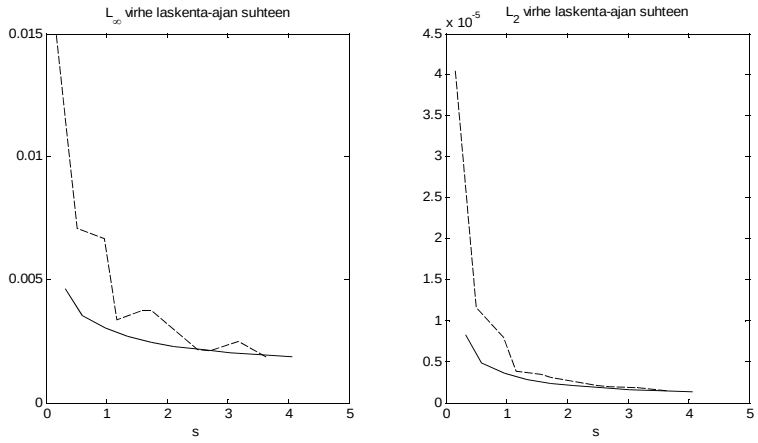
Kuva 7: Pienin etäisyys pisteisiin (0.25,0.25) ja (0.75,0.75) yksikköneliössä. Vasemmalta oikealle: Tarkka, vanhan menetelmän ja uuden menetelmän ratkaisu.



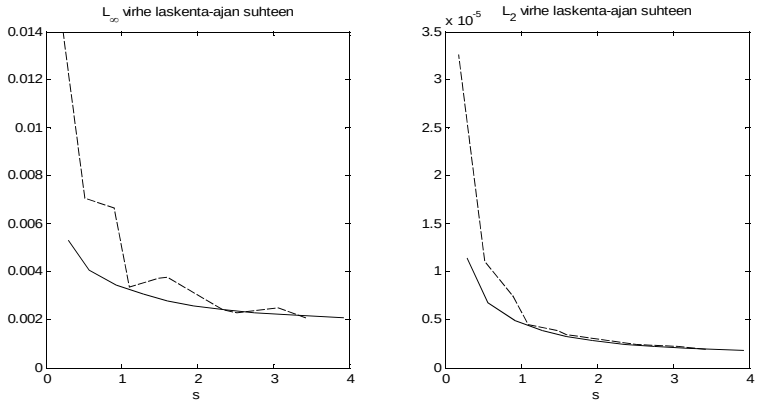
Kuva 8: Laskenta-aika, L_∞ ja L_2 virheet verkon koon N funktiona laskettaessa etäisyyttä pisteisiin (0.25,0.5) ja (0.75,0.5). Yhtenäinen viiva: vanha menetelmä. Katkoviiva: uusi menetelmä.



Kuva 9: Laskenta-aika, L_∞ ja L_2 virheet verkon koon N funktiona laskettaessa etäisyyttä pisteisiin (0.25,0.25) ja (0.75,0.75). Yhtenäinen viiva: vanha menetelmä. Katkoviiva: uusi menetelmä.



Kuva 10: Virheet laskenta-ajan funktiona laskettaessa etäisyyttä pisteisiin (0.25,0.5) ja (0.75,0.5). Yhtenäinen viiva: vanha menetelmä. Katkoviiva: uusi menetelmä.



Kuva 11: Virheet laskenta-ajan funktiona laskettaessa etäisyyttä pisteisiin (0.25,0.25) ja (0.75,0.75). Yhtenäinen viiva: vanha menetelmä. Katkoviiva: uusi menetelmä.

2.3 Pistemäisen valonlähteen varjoalueet

Seuraavaksi uutta menetelmää kokeillaan etsittäessä pistemäisen valonlähteen aiheuttamia varjoalueita. Sethian (1998, p. 49) esitteli, kuinka nopeita marssimenetelmiä voidaan käyttää varjoalueiden etsimiseen. Ideana on laskea etäisyys valonlähteestä kuhunkin pisteeseen kahteen kertaan; seinillä ja ilman seiniä. Mikäli etäisyys pisteeseen on suurempi, kun otetaan seinät huomioon, joudutaan pisteeseen saapumiseksi kiertämään esteitä ts. valonlähteestä ei ole suoraa reittiä pisteeseen, joten piste on varjossa.

Käytännössä ratkaistaan kaksi tehtävää

a) $$|\nabla u(x)|=1 \quad u(v)=0,\ v\in\Omega \tag{14}$$

b) $$|\nabla u(x)|=\begin{cases}\infty & ,kun\ x\in\Pi\subset\Omega\\ 1 & ,muulloin\end{cases} \quad u(v)=0,\ v\in\Omega \tag{15}$$

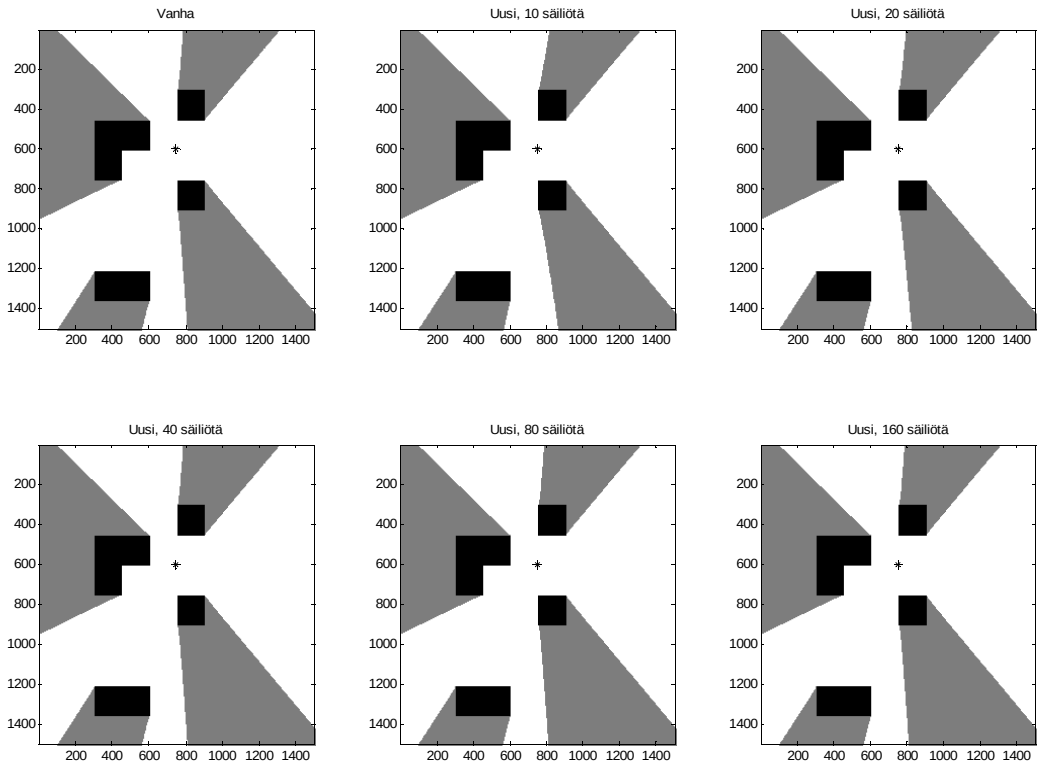
missä Π on seiniä vastaava alue ja v on piste, jossa on valonlähde. Merkataan a)-kohdan ratkaisua $u_{ei\ seiniä}(x)$ ja b)-kohdan ratkaisua $u_{seinät}(x)$. Jos $u_{seinät}(x)>u_{ei\ seiniä}(x)$, piste on varjossa.

Käytännössä numeeristen epätarkkuuksien takia joudutaan hyväksymään pienet virheet ts. piste on varjossa jos $u_{seinät}(x) > u_{ei\ seiniä}(x) + \epsilon$. Tässä on käytetty $\epsilon = 1$.

Kokeillaan eri menetelmiä tehtävässä, joissa $\Omega = [0,1500]\,x\,[0,1500]$, $h = 1$ ja Π on muutaman satunnaisen suorakaiteen unioni. Kuvassa 12 on esitetty seinät, varjot ja valaistut alueet sekä vanhalla että uudella menetelmällä ratkaistuna eri säiliöiden lukumäärällä *d*.

Huomataan, että jo pienellä määrällä säiliöitä saadaan lähes identtinen lopputulos alkuperäisen menetelmän kanssa. Tässä tapauksessa tarkoilla u :n arvoilla kussakin pisteessä ei ole niinkään väliä; riittää, että u :n arvot ovat isompia, kun pisteestä ei ole näköyhteyttä valonlähteeseen.

Taulukossa 1 on esitetty laskenta-aika ja uuden menetelmän antamien virheellisten pikseleiden lukumäärä, verrattuna vanhan antamaan tulokseen. Laskenta-aika vanhaan menetelmään nähden on kaikissa tapauksissa noin puolet. Lisäksi huomataan, että säiliöiden määrä on jokaisessa tapauksessa vielä paljon pienempi kuin joukossa *lähellä* olevien solmujen määrä, koska laskenta-aika ei juurikaan kasva säiliöiden lukumäärää lisättäessä. Viimeisessä tapauksessa $d = 160$ laskenta-aika on edelleen noin puolet vanhan menetelmän laskenta-ajasta, mutta lopputulos on täysin identtinen.



Kuva 12: Varjojen laskeminen vanhalla ja uudella menetelmällä, käyttäen eri määrää säiliöitä O(N) versiossa. Valonlähde on merkattu tähdellä, valkoiset alueet ovat valaistuja, mustat esteitä ja harmaat varjoja.

Taulukko 1: Varjojen laskeminen vanhalla ja uudella menetelmällä. Virheellisten pikseleiden lukumäärä on verrattuna vanhan menetelmän antamaan tulokseen. Arvot 10 ajokerran keskiarvoja.

	Vanha	$d=10$	$d=20$	$d=40$	$d=80$	$d=160$
Laskenta-aika (s)	19.82	11.30	11.30	11.33	11.38	11.74
Virheellisiä pikseleitä	--	21556	6052	520	4	0

2.4 Robotin reitinetsintä

Viimeisessä kokeessa kokeillaan nopeita marssimenetelmiä reitinetsintätehtävässä. Robotti on $1500\,x\,1500$ kartalla, pisteessä $(100,100)$ ja haluaa mennä lyhyintä reittiä pisteeseen $(1400,1400)$. Kartalla on esteitä, joille $f=\infty$, ja suorakaiteen muotoisia alueita, joiden hitaudet f arvotaan tasajakaumasta väliltä $[1,2]$. Tehtävänä on siis ratkoa

$$\begin{aligned} &|\nabla u(x)| = \begin{cases} \infty & ,kun\ x \in \Pi \subset \Omega \\ c(x) & ,muulloin \end{cases} \\ &u(r)=0,\ r\in\Omega \end{aligned} \qquad (16)$$

missä Π on esteitä vastaava alue ja r on robotin lähtöpiste, $r=(100,100)$. Kuljettavien maastojen hitaudet ovat funktiossa $c(x)$, $1\le c(x)\le 2$.

Ensin tehtävä ratkaistaan nopeilla marssimenetelmillä ($h=1$) josta saadaan u. Tämän jälkeen lähdetään loppupisteestä $(1400,1400)$, kulkien aina laskevan gradientin suuntaan. Integrointi tehdään Heunin menetelmällä, joka on (negatiivinen merkki, koska mennään laskevan gradientin suuntaan):

$$\begin{aligned} &x_{tmp} = x(k) - h\nabla u(x(k)) \\ &x(k+1) = x(k) - h/2(\nabla u(x(k)) + \nabla u(x_{tmp})) \end{aligned} \qquad (17)$$

Integroinnissa otettiin askelpituudeksi $h=1$, gradienttien approksimointiin käytetään samaa ylöspäin approksimaatiota kuin marssimenetelmässä, $x_1(0)=1400$, $x_2(0)=1400$ ja integrointi lopetetaan, kun $(x_1-100)^2+(x_2-100)^2<1$ ts. kun ollaan yhden ruudun päässä robotin lähtöpisteestä. Kun algoritmin löytämä reitti C on laskettu, saadaan reitin kesto viivaintegraalilla $\int_C f\,ds$. Tämä lasketaan numeerisesti käyttäen trapetsimenetelmää.

Kuvassa 13 on esitetty eri säiliömäärillä saadut tulokset.

Taulukossa 2 on esitetty tehtävän ratkomiseen kulunut aika ja menetelmän löytämän reitin kesto. Uusi menetelmä on selvästi vanhaa nopeampi, mutta sen löytämien reittien kestot ovat käytännössä identtisiä vanhan menetelmän löytämän reitin keston kanssa. Uusi menetelmä antoi nyt jo todella pienellä määrällä säiliöitä melkein vastaavia tuloksia vanhan menetelmän kanssa.

3 Yhteenveto ja jatkotutkimukset

Kaikissa kokeissa uusi menetelmä oli antoi lähes yhtä hyviä tuloksia kuin vanha menetelmä. Käytännön kokeissa uusi menetelmä oli selkeästi vanhaa menetelmään nopeampi. Nopeudestaan huolimatta uusi menetelmä on jopa yksinkertaisempi implementoida kuin vanha, koska epäsiistissä prioriteettijonossa säiliön etsiminen vaatii vain yksinkertaista aritmetiikkaa, kun taas normaalissa prioriteettijonossa joudutaan tekemään paljon operaatioita, jotta keko säilyisi kekona alkion

lisäyksen, poiston tai muutoksen yhteydessä.

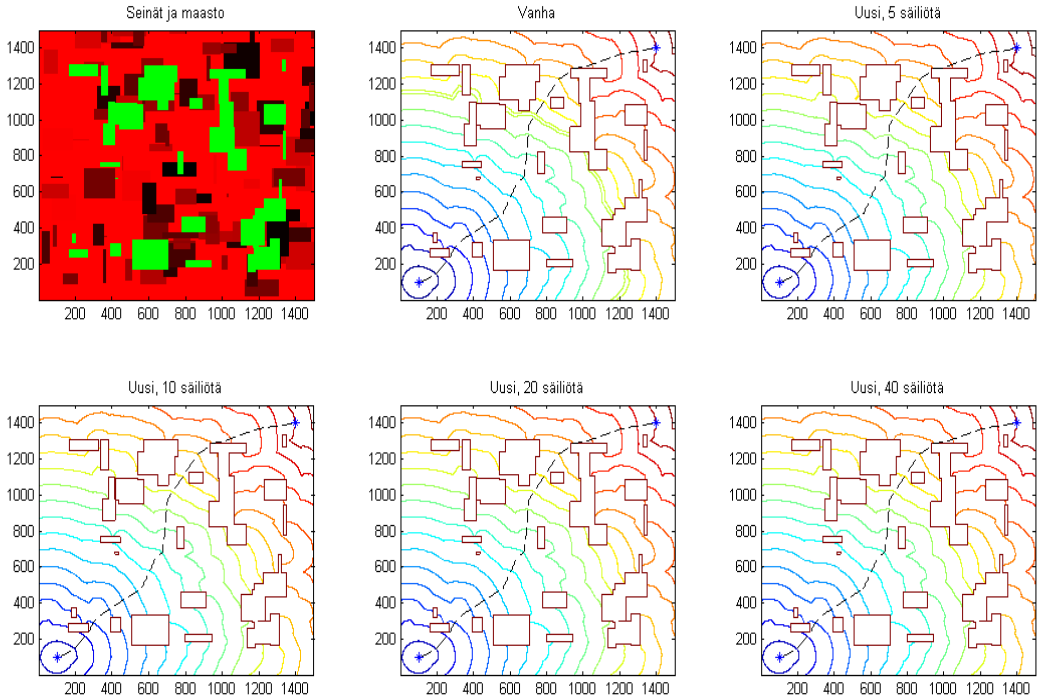
Käytännön sovelluksissa algoritmi on niin nopea, että sitä voidaan harkita käytettävän jopa reaaliaikasovelluksissa. Algoritmi tullaan epäilemättä implementoimaan myös grafiikkakorteilla, mahdollistaen sovellukset esimerkiksi peleissä ja konenäössä. Toisaalta myös resurssirajoitteiset laitteet (mobiilit robotit, kännykät) hyötyvät uudesta menetelmästä.

Uusi menetelmä ei kuitenkaan paranna muistivaatimuksia. Algoritmin vaatima muistitila on edelleen $O(N)$. On vain muutamia tapoja parantaa tätä:

Yksi vaihtoehto on siirtyä strukturoimattomaan verkkoon, käyttäen suurta määrää solmuja siellä, missä tarvitaan tarkkuutta. Olisi mielenkiintoista nähdä, miten uusi menetelmä menestyy tällaisissa tapauksissa. Verkon adaptiivisuuskin on mahdollista.

Toinen vaihtoehto soveltuu vain tilanteisiin, joissa funktiota u ei tarvitse tietää kerralla. Algoritmin luonteesta johtuen kun u:n arvot on laskettu alueessa, joka ei ole enää kosketuksissa mihinkään joukon *lähellä* pisteisiin, alueen arvoilla ei ole enää mitään vaikutusta myöhemmin laskettaviin u:n arvoihin. Näin voidaan käyttää tässä muistitilassa sijaitsevat arvot lopputuloksen laskemiseksi ja sen jälkeen vapauttaa muistitila käytöstä.

Tulevaisuudessa uutta menetelmää olisi syytä kokeilla gradientin korkeamman kertaluvun aproksimaatioilla (Sethian 1999, s. 26), vapaalla verkolla (Sethian ja Vladimirsky 2000) tai napakoordinaatistossa (Alkhalifah ja Fomel 2001). Perusidea (prioriteettijonon korvaaminen epäsiistillä prioriteettijonolla) on suoraan sovellettavissa kaikkiin em. tapauksiin. Gradientin korkeamman kertaluvun aproksimaatioiden yhteydessä voisi olettaa, että säiliöiden määrä joudutaan valitsemaan eri tavalla, jotta valitsemisjärjestyksestä johtuva virhe saadaan pysymään tehtävän diskretoinnista johtuvaa virhettä vastaavana.



Kuva 13: Robotin reitin etsiminen vanhalla ja uudella menetelmällä, käyttäen eri määrää säiliöitä. Vasen yläkulma: Kuljettava maasto. Vihreä on seinää, punainen kuljettavaa maastoa. Kirkkaampi punainen on hitaampaa maastoa. Loput kuvat: Eri menetelmien löytämät reitit.

Taulukko 2: Robotin reitin etsiminen vanhalla ja uudella menetelmällä, käyttäen eri määrää säiliöitä (parametri d). Taulukossa annettuna algoritmin käyttämä aika sekä löytämän reitin kesto.

	Vanha	$d=5$	$d=10$	$d=20$	$d=40$
Laskenta-aika (s)	8.68	4.33	4.33	4.35	4.37
Reitin kesto	3.2030	3.2084	3.2047	3.2033	3.2035

Viitteet

Alkhalifah, T., Fomel, S., 2001. Implementing the fast marching eikonal solver: spherical versus Cartesian coordinates. *Geophysical Prospecting*, 49 (2), s. 165–178.

Alouges, F., Desimone, A., 1999. Plastic Torsion and Related Problems. *Journal of Elasticity,* 55, pp. 231–237.

Garrido, S., Moreno, L., Abderrahim, M., Martin, F., 2006. Path Planning for Mobile Robot Navigation using Voronoi Diagram and Fast Marching. *Proceedings of the 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*. s. 2376–2381

Sethian, J. A., 1996. A fast marching level set method for monotonically advancing fronts. *Proceedings of the National Academy of Sciences*, 93 (4), s. 1591–1595.

Sethian, J.A., 1999. Fast Marching Methods. *SIAM Review*, 41 (2), s. 199–235.

Sethian, J. A., Vladimirsky, A., 2000. Fast methods for the Eikonal and related Hamilton–Jacobi equations on unstructured meshes. *Proceedings of the National Academy of Science.* 97 (11), s. 5699–5703.

Sofou, A., Maragos, P., 2003. PDE-based modeling of image segmentation using volumic flooding. *Proceedings of 2003 International Conference on Image Processing.* 2, s. 431–4.

Yatziv, L., Bartesaghi, A., Sapiro, G., 2006. O(N) implementation of the fast marching algorithm. *Journal of Computational Physics*, 212, s. 393–399.