TiRaLabra, toteutusdokumentti

Arto Kaikkonen

Ohjelman yleisrakenne

Ohjelma koostuu ReitinEtsija-luokkaa laajentavista luokista, joista kukin on yhden reitinhakualgoritmin toteutus. Jokaisella luokalla on konstruktori, joka saa parametrinaan XYVerkko-olion. Tämän jälkeen kutsutaan etsiLyhinReitti()-metodia, jolloin algoritmi suoritetaan. Tämän jälkeen voidaan hakea getterillä lyhin reitti, sen pituus sekä algoritmista riippuen myös muita arvoja.

XYVerkko koostuu taulukoista, joihon on tallennettu verkon solmut (XYKoordinaatti-oliot) sekä solmujen väliset etäisyydet. Tärkein konstruktori on konstruktori, jolle annetaan parametreina solmujen määrä ja alueen koko (x- ja y-koordinaattien maksimiarvot), ja joka arpoo verkon näiden parametrien mukaisesti. XYKoordinaatti taas on luokka, jonka olioilla on muuttujinaan niiden kuvaaman pisteen x- ja y-koordinaatit.

Pinon toteutin pinoalkion avulla. Pinoon voi tallentaa mitä tahansa olioita. Pino tuntee päällimmäisen pinoalkion, joka taas tuntee sen sisältämän olion sekä seuraavan pinoalkion.

Saavutetut aika- ja tilavaativuudet

Tässä toteutuksessa on oletettu verkon olevan täydellinen. Koska kaikista solmuista lähtee kaari kaikkiin muihin solmuihin, on kaarien määrä O-analyysin näkökulmasta solmujen määrän neliö. Näin ollen en ole huomioinut kaarien määrää O-analyysiä tehdessäni.

AntSystem

AntSystem-algoritmin aikavaatimus on n*|V|^2, missä n on iteraatiokierrosten määrä ja V solmujen määrä. Algoritmin palauttaman tuloksen tarkkuus riippuu kuitenkin iteraatiokierrosten määrästä siten, että hyvän tuloksen vaatima kierrosmäärä on riippuvainen solmujen (tai ehkä ennemminkin kaarien) määrästä. Näin ollen aikavaativuus on ainakin V^3 (ehkä jopa V^4) mikäli halutaan, että tulokset säilyvät vertailukelpoisina.

AinaLahimpaan

AinaLahimpaan-algoritmi toimii ajassa V^3. Metodissa etsiLyhinReitti kutsutaan V kertaa metodia aloitaEtsiminen. Metodissa aloitaEtsiminen kutsutaan V kertaa metodeja ollankoLopussa() ja lahinKaymaton(), jotka toimivat molemmat ajassa O(V). Lisäksi kutsutaan kerran ajassa V toimivaa paivitaLyhinReitti-algoritmia.

Mikäli metodi aloitaEtsiminen suoritettaisiin vain kerran, toimisi algoritmi ajassa O(V^2). Tällöin lopputulos riippuisi siitä mistä solmusta aloitetaan, eikä algoritmi antaisi samalla verkolla aina samaa tulosta.

BruteForce

BruteForce-algoritmi käy läpi kaikki mahdolliset reitit, joten sen aikavaativuus on O(V!). Tilavaativuus on O(V), sillä jokaisen metodin tilavaativuus on O(1) ja rekursiopinon korkeus on korkeintaan V.

BranchAndBound & BranchAndBound2

BranchAndBoundien pahimman tapauksen aikavaativuus on sama kuin BruteForcella, eli O(V!). Käytännössä BranchANdBound-algoritmit toimivat nopeammin, mutta O-analyysin näkökulmasta aikaavaativuus on sama.

KarpHeld

KarpHeldin aikavaativuuden arviointiin eivät omat kykyni riitä, mutta aiheesta kirjoitettujen artikkeleiden mukaan se on O(V^2 * 2^V).

Suorituskyky- ja O-analyysivertailu

BruteForce vs BranchAndBoundit

Vertailin BruteForcen ja BranchAndBound-algoritmien toimintaa ainoastaan pienillä syötteillä, sillä tietokoneeni tehot loppuivat kesken jo hieman yli kymmenen solmun verkoilla. Kaikki kolme algoritmia löytävät aina lyhimmän reitin, ja vertailu kohdistui ainoastaan algoritmien nopeuteen. Ajoin algoritmit 100 erilaisella 10 ja 11 solmun verkolla sekä 10 erilaisella 12 solmun verkolla, tulosten keskiarvot löytyvät oheisesta taulukosta. Mittaamiseen käytin javan System.currentTimeMillis()-metodia. Mittausten perusteella BranchAndBound-algoritmit toimivat lähes yhtä nopeasti ja BruteForce selkeästi hitaammin. Suhteellinen ero BruteForcen ja BranchAndBound-algoritmien välillä näyttäisi kasvavan syötteen koon kasvaessa. Oletettavasti BranchAndBoundit jättävät käymättä läpi sitä isomman osan reiteistä, mitä enemmän reittejä on.

		BruteForce	BranchAndBound1		BranchAndBound2	
solmuja	otoskoko	Aika (ms)	Aika (ms)	% BruteForcesta	Aika (ms)	% BruteForcesta
10	100	54,76	9,36	17,1 %	9,1	16,6 %
11	100	591,22	42,79	7,2 %	42,96	7,3 %
12	10	6734,6	257,6	3,8 %	262,7	3,9 %

Karp-Held vs BranchAndBound

Karp-Held eroaa muista algoritmeista siten, että se ei löydä aina Hamiltonin kierrosta lainkaan. Sen sijaan jos se löytää kierroksen, on tämä kierros varmuudella lyhin Hamiltonin kierros. KarpHeld-algoritmi perustuu muuttuvien ”valepainojen” käyttöön. Jokaisella iteraatiokierroksella valepainojen arvoja muutetaan hieman ja uusien painojen avulla haetaan pienin virittävä l-puu. Mikäli tämä l-puu on Hamiltonin kierros, on se samalla lyhin Hamiltonin kierros. Mikäli se ei ole Hamiltonin kierros, on l-puun kaarien yhteenlaskettu paino alaraja lyhimmän Hamiltonin kierroksen pituudelle. Koska Hamilton kierrosta ei aina löydetä, on iteraatiokierrosten maksimiarvo asetettava etukäteen, jotta algoritmi ei jää ikuiseen looppiin. Algoritmi laskee iteraatiokierrosten määrää ja

nollaa laskurin aina kun löydetty virittävä l-puu on painavampi kuin painavin aiemmin löydetty l-puu.

Testasin KarpHeldiä ja BranchAndBound algoritmia tuhannella 11 solmun verkolla. Valitsin ensin iteraatiokierrosten määräksi 3, 30, 300 ja 3000, mutta tulokset 30, 300, ja 3000 kierroksen osalta olivat käytännössä täysin identtiset. Näin ollen vaihdoin määriksi 4, 8, 16 ja 32.

Suorituskykyeroa BranchAndBoundiin ei saanut järkevästi mitattua, sillä pienillä verkoilla Karp-Held suoriutui tehtävästään keskimäärin alle millisekuntissa. Oheisessa taulukossa on eriteltynä tulokset niiltä osin kun Karp-Held ei löytänyt oikeaa vastausta.

v = 11	**Iteraatiokierroksia**			
n=1000	**4**	**8**	**16**	**32**
Vastausta ei löydy	20,6 %	6,9 %	4,9 %	4,6 %
Ero keskimäärin*	0,3 %	0,2 %	0,2 %	0,2 %
Maksimiero	2,5 %	1,8 %	1,0 %	1,0 %

* Jakajana niiden tapausten määrä, joissa vastausta ei löydy

Karp-Held 10-50 solmun verkoilla

Kokeilin Karp-Held –algoritmia erikokoisilla verkoilla ja eri iteraatiokierrosten lukumäärillä. Empiiristen kokeiden perusteella näyttäisi siltä, että iteraatiokierrosten määrän kasvattaminen rajattomasti ei paranna algoritmin toimintaa lainkaan. Tulokset 10-50 solmun verkoilla olivat täsmälleen samat riippumatta siitä oliko iteraatiokierroksia 256 vai 65536. Näyttäisi siltä, että Karp-Held ei suuremmilla verkoilla anna kovinkaan usein oikeaa vastausta. En keksinyt luotettavaa tapaa mitata Karp-Heldin antaman lyhimmän reitin pituuden alarajan hyvyyttä suuremmilla verkoilla, mutta oletettavasti arviot ovat melko tarkkoja. Tästä syystä käytän Karp-Heldiä arvioidessani lyhimmän kierroksen pituutta suuremmilla verkoilla.

Oikeiden vastausten osuus

n=1000	**Iteraatiokierroksia**			
solmuja	**4**	**16**	**256**	**65536**
10	82,9 %	97,8 %	97,8 %	97,8 %
20	36,4 %	72,7 %	75,1 %	75,1 %
30	10,8 %	39,5 %	47,2 %	47,2 %
40	3,6 %	18,4 %	29,2 %	29,2 %
50	1,8 %	11,7 %	20,5 %	20,5 %

Keskimääräinen arvio lyhimmän reitin pituudesta

n=1000	**Iteraatiokierroksia**			
solmuja	**4**	**16**	**256**	**65536**
10	100	100,031	100,031	100,031
20	100	100,460	100,469	100,469
30	100	100,835	100,885	100,885

40	100	100,995	101,116	101,116
50	100	100,998	101,162	101,162

AntSystem ja AinaLahimpaan

AntSystemin toiminta riippuu sille annetuista parametreista. Kohtuullisen hyvillä parametreilla AntSystem näyttäisi antavan keskimäärin muutaman prosentin paremman tuloksen kuin AinaLahimpaan. Molempien aikavaativuus on alla O(V^3), sillä määrittelin AntSystemin hakujen määräksi 10*V. AntSystem on kuitenkin huomattavasti raskaampi, joten se on reilusti AinaLahimpaan-algoritmia hitaampi. Lisäksi AntSystem perustuu osittain sattumaan, joten teoriassa sen palauttama reitti voi olla todella huonokin.

Keskimääräinen ero arvioituun lyhimpään reittiin

v	AinaLahimpaan	AntSystem
30	9 %	6 %
60	13 %	9 %
90	15 %	12 %
120	16 %	12 %

Maksimiero

v	AinaLahimpaan	AntSystem
30	18 %	18 %
60	28 %	17 %
90	26 %	22 %
120	25 %	23 %

Keskimääräinen kesto (ms)

v	AinaLahimpaan	AntSystem
30	0	40
60	1	319
90	3	1 064
120	6	2 502

Johtopäätökset

Tutkimistani algoritmeista tarkan vastauksen antavat algoritmit ovat käyttökelpoisia vain melko pienillä verkoilla. Karp-Held –algoritmi toimii vielä hieman isommilla verkoilla, mutta mitä isompi verkko on, sitä todennäköisemmin se ei palauta lainkaan Hamiltonin kierrosta.

Karp-Held on kuitenkin hyödyllinen kun arvioidaan algoritmien antamien vastausten tarkkuutta. Karp-Held palauttaa näet melko tarkan arvion lyhimmän kierroksen pituudesta. Tätä pituutta voidaan verrata muiden algoritmien palauttamien kierrosten pituuksiin.

Aina lähimmän käymättömän naapurin valitseva algoritmi on kohtuullisen nopea suurillakin syötteillä. Sen palauttama reitti voi kuitenkin varsinkin suurilla syötteillä olla kymmeniä prosentteja pidempi kuin optimi. AntSystem-algoritmin avulla voidaan parantaa tulosta muutaman prosentin, mutta se on huomattavasti raskaampi.

Työn mahdolliset puutteet ja parannusehdotukset

Vaikka algoritmeja tuskin saa hiottua nopeammiksi O-analyysimielessä, voi niiden toimintaa varmasti tehostaa paljonkin. Samoin kaikkien algoritmien testaaminen todella isoilla verkoilla ei ollut välineilläni mahdollista.

AntSystem-algoritmin parametrien vaikutusta lopputulokseen voisi tutkia enemmän. Löysin kokeilemalla kohtuullisen hyvät parametrit tietyn kokoisille verkoille, mutta ajan ja laskentatehon puutteen takia en pystynyt kovin kattavia testejä tekemään.

Lähteet

Dorigo, Maniezzo, Colorni: Ant System: Optimization by a Colony of Cooperating Agents (IEE, 1/1996)

Ruohonen Keijo: Graafiteoria (http://math.tut.fi/~ruohonen/GT.pdf)

http://en.wikipedia.org/wiki/Travelling_salesman_problem