Pretty Simple TSP

Määrittelydokumentti

Jarno Leppänen

20.1.2013

1 Johdanto

Pretty simple TSP (`pstsp`) on approksimatiivisen algoritmin toteutus kauppamatkustajan ongelmalle.

1.1 Teoriaa

Symmetrisen verkon pienin virittävä puu on luonteva alaraja kauppamatkustajan ongelman ratkaisulle, sillä minkä tahansa kaaren poistaminen ratkaisusta tuottaa Hamiltonin polun, joka on samalla eräs verkon virittävä puu [13]. Jos verkko on lisäksi täydellinen ja sen kaaret toteuttavat kolmioepäyhtälön, eli suora reitti solmusta a solmuun b on aina lyhyempi kuin reitti solmun c kautta ($d_{ab} < d_{ac} + d_{cb}$), voidaan laatia pienimpään virittävään puuhun perustuva algoritmi, jonka tuottama ratkaisu on korkeintaan vakiokertoiminen optimaaliseen ratkaisuun verrattuna:

1. Etsitään verkolle G pienin virittävä puu T.
2. Lisätään verkkoon H puun T kaaret ja jokaista puun T kaarta (u, v) vastaava kaari (v, u).
3. Etsitään jokaisen kaaren täsmälleen kerran läpikäyvä *Eulerin polku* C verkossa H. Polun pituus on kaksi kertaa puun T pituus.
4. Laaditaan kaikki verkon G solmut täsmälleen kerran läpikäyvä *Hamiltonin sykli* kulkemalla polun C kaaria pitkin ja hyppäämällä yli kaikki jo läpikäydyt solmut.

Koska verkon kaaret toteuttavat kolmioepäyhtälön, on kohdassa 4 tuotettu sykli lyhyempi kuin polku C. Algoritmi siis tuottaa kauppamatkustajan ongelmaan ratkaisun, joka on korkeintaan kaksi kertaa optimaalista ratkaisua pidempi.

Ratkaisu voidaan myös tuottaa yhdistämällä kaarilla sellaiset pienimmän virittävän puun solmut, joiden asteluku on pariton luku ja muodostamalla Eulerin polku syntyneessä verkossa. *Christofidesin algoritmissa*[8] yhdistävät kaaret valitaan siten, että niiden painojen summa on mahdollisimman pieni. Voidaan osoittaa, että näin saadun ratkaisun pituuden yläraja on 3/2 kertaa optimiratkaisun pituus.

2 Toteutettavat algoritmit

`pstsp` koostuu kahdesta osasta.

2.1 Lyhimpien polkujen haku kaikkien verkon solmujen välillä

Ensimmäisessä osassa syötteenä oleva verkko käsitellään siten, että esiteltyä algoritmia voidaan käyttää. Jos verkko ei ole täydellinen tai toteuta kolmioepäyhtälöä, laaditaan uusi verkko hakemalla lyhimmät polut kaikkien solmujen välillä ja asettamalla nämä kaarien painoiksi. Kaikkien lyhimpien polkujen hakuun toteutetaan Floyd-Warshallin algoritmi[10] ja harvoille verkoille paremmin soveltuva Johnsonin algoritmi[11], jos aikaa jää.

2.2 Approksimatiivinen algoritmi kauppamatkustajan ongelmalle

Toinen osa koostuu approksimatiivisen algoritmin toteutuksesta kauppamatkustajan ongelmalle. Pienimmän virittävän puun etsintään toteutetaan Primin algoritmi[12] ja Hamiltonin sykli etsitään yksinkertaisella syvyyssuuntaisen haun muunnoksella. Jos aikaa jää, toteutetaan Christofidesin algoritmi. Tässä tarvittavan parittoman asteluvun solmujen täydellisen sovittamisen etsintään on olemassa polynomisessa ajassa toimiva *blossom-algoritmi*[7], joka on tosin erittäin haastava toteutettaa.

3 Motivaatio

Esitelty algoritmi soveltuu hyvin toteutettavaksi tietorakenteiden ja algoritmien perusteiden opettelun jälkeen. Algoritmissa ja verkon esikäsittelyssä hyödynnetään monia kurssilla esiteltyjä menetelmiä, kuten lyhimpien polkujen ja pienimmän virittävän puun etsintää.

Käytännössä kauppamatkustajan ongelmaan on olemassa heuristiikkoja, jotka löytävät nopeammin parempia ratkaisuja, kuin tässä työssä käytettävät algoritmit.[13] Suurinkiin ongelmiin voidaan lisäksi löytää eksakti ratkaisu kokonaislukuoptimoinnin menetelmillä.

4 Laskennan vaativuus

Työn ensimmäisen osa on toista osaa vaativampi laskennallisesti. Floyd-Warshallin algoritmin aikavaativuus on $\Theta(|V^3|)$ ja tilavaativuus $\Theta(|V^2|)$. Binäärikekoa hyödyntävää Dijkstran algoritmia[9] käyttävän Johnsonin algoritmin pahimman tapauksen aikavaativuus on luokkaa $\mathcal{O}((|V| + |E|)|V| \log |V|)$ ja tilavaativuus $\mathcal{O}(|V|)$.

Binäärikekoa hyödyntävän Primin algoritmin pahimman tapauksen aikavaativuus on luokkaa $\mathcal{O}((|V| + |E|) \log |V|)$ ja tilavaativuus $(|V|)$. Christofidesin algoritmin aikavaativuus on laskennallisesti vaativan blossom-algoritmin vuoksi jopa $\mathcal{O}(|V|^4)$.

5 Ohjelman rakenne

Työ koostuu C++-header-kirjastosta, jossa on toteutettu vaadittavat algoritmit, sekä kirjastoa hyödyntävästä komentoriviltä käytettävästä ohjelmasta. Syötteenä hyväksytään `TSPLIB`-formaatissa[2] ja AI Odyssey -kurssilla[4] käytetyssä formaatissa olevia tiedostoja. Ohjelma tunnistaa, tarvitseeko syötteenä oleva verkko käsittelyä ja keskeyttää ajon, jos verkko sisältää negatiivisia syklejä. Projektinhallintaan käytetään `cmake`-työkalua[5], yksikkötestaukseen `boost test`-kirjastoa[3] ja dokumentointiin `doxygen`-työkalua[1]. Paperidokumenttien laatimiseen käytetään LaTeX-ladontajärjestelmää[6].

Viitteet

[1] Dimitri van Heesch. Doxygen. URL `http://www.stack.nl/~dimitri/doxygen/`.

[2] Gerhard Reinelt. TSPLIB. URL `http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/`.

[3] Google. Google Test. URL `http://code.google.com/p/googletest/`.

[4] Jarmo Isotalo. AI Odyssey — Challenge1. URL `https://github.com/AIOdyssey/wiki/wiki/Challenge1`.

[5] Kitware. CMake. URL `http://www.cmake.org`.

[6] Leslie Lamport. LaTeX. URL `http://www.latex-project.org/`.

[7] Wikipedia. Blossom algorithm — Wikipedia, The Free Encyclopedia, . URL `http://en.wikipedia.org/wiki/Blossom_algorithm`.

[8] Wikipedia. Christofides algorithm — Wikipedia, The Free Encyclopedia, . URL `http://en.wikipedia.org/wiki/Christofides_algorithm`.

[9] Wikipedia. Dijkstra's algorithm — Wikipedia, The Free Encyclopedia, . URL `http://en.wikipedia.org/wiki/Dijkstra's_algorithm`.

[10] Wikipedia. Floyd–Warshall algorithm — Wikipedia, The Free Encyclopedia, . URL `http://en.wikipedia.org/wiki/Floyd-Warshall_algorithm`.

[11] Wikipedia. Johnson's algorithm — Wikipedia, The Free Encyclopedia, . URL `http://en.wikipedia.org/wiki/Johnson's_algorithm`.

[12] Wikipedia. Prim's algorithm — Wikipedia, The Free Encyclopedia, . URL `http://en.wikipedia.org/wiki/Prim's_algorithm`.

[13] Wikipedia. Travelling salesman problem — Wikipedia, The Free Encyclopedia, . URL `http://en.wikipedia.org/wiki/Travelling_salesman_problem`.