Toteutusdokumentti

Ohjelman yleisrakenne

Ohjelma on jaettu:

- Käyttöliittymään (Toimitusreitinlaskijasovellus-luokka)
- Karttapalveluihin (GoogleMaps-luokka) (Hyödyntää Google Distance Matrix API Java-clientia)
- Algoritmeihin (Pääalgoritmit KauppamatkustajaBruteForce-luokka, KauppamatkustajaDynaaminen-luokka ja KauppamatkustajaHeuristinen-luokka. Lisäksi apualgoritmit Reitinpituus-luokka ja ReittienVertailija-luokka joita käytetään sanallisten analyysien tuottamiseen käyttäjälle pääalgoritmien toiminnasta).

Käyttöliittymä on toteutettu nettisivun (HTML+CSS+Javascript) avulla, joka näytetään käyttäjälle Spark Javan ja Thymeleafin avulla. Nettisivulla kartta näytetään käyttäjälle Google Maps JavaScript API:lla, käyttäjän syöttämien osoitteiden ja paikkojen selventämiseen käytetään Google Maps JavaScript API:n Places kirjaston toimintoa ”Autocomplete for Addresses and Search Terms” ja reitin piirtämiseen kartalle käytetään Google Maps JavaScript API:n Directions-palvelua.

Saavutetut aika- ja tilavaativuudet O-analyysin mukaan

Saavutettu Brute-forcella ja Dynaamisella algoritmilla tavoitellut aikavaativuudet. Heuristiselle algoritmille ei asetettu aikavaativuustavoitetta etukäteen. Tilavaativuustavoitteita ei asetettu ollenkaan.

Brute-force toteutuksella saavutettu aikavaativuus O(n!). Tämä on selvää koska algoritmi käy läpi kaikki mahdolliset reittivaihtoehdot ja reittivaihtoehtoja on selvästikkin n! kappaletta. Ennen algoritmin rekursiota ja rekursion jälkeen suoritetut operaatiot ovat aikavaativuudeltaan O(n) joten nämä operaatiot eivät vaikuta algoritmin kokonaisaikavaativuuteen.
Brute-force toteutuksella saavutettu tilavaativuus $O(n^2)$, koska algoritmi toimii branch-and-bound taktiikalla, eli vain se ”puun” haara on muistissa jota ollaan tutkimassa. Koska puu on korkeudeltaan n ja koska jokaisella tasolla tarvitsee pitää muistissa missä solmuissa on vierailtu ja milloin, niin täten tilavaativuudeksi tulee $O(n^2)$.

Dynaamisella toteutuksella saavutettu aikavaativuus $O(n^2 2^n)$ [1], koska toteutettu Held-Karp algoritmi [2]. Algortimin ytimen jälkeen suoritetut operaatiot, luoReittiOhjeet($O(n^2)$) ja muunnaReittiOhjeet(O(n)), eivät vaikuta algoritmin kokonaisaikavaativuuteen.
Dynaamisella toteutuksella saavutettu tilavaativuus $O(n2^n)$ [1], koska toteutettu Held-Karp algoritmi [2]. Sivuoperaatiot eivät selvästikään yllä tuohon tilavaativuuteen joten ne eivät vaikuta kokonaistilavaativuuteen.

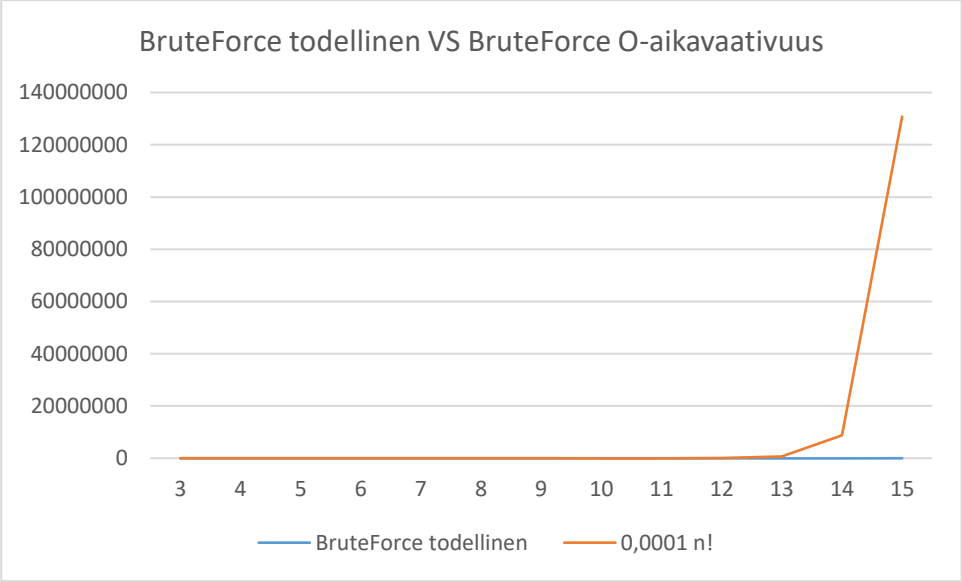
Heuristisella toteutuksella saavutettu aikavaativuus $O(n^2)$, koska algoritmissa on pääluuppi joka suoritetaan n kertaa(lisää aina yhden uuden solmun reittiin) ja pääluupin sisällä on 2 luuppia jotka suoritetaan molemmat n kertaa (selvitetään aina mikä solmu on lähinnä tähän astisen reitin päätä ja mikä solmu on lähinnä häntää).
Heuristisella toteutuksella saavutettu tilavaativuus O(n), koska algoritmi luo n-kokoisen vierailtu-taulukon jolla se pitää kirjaa missä solmuissa on jo vierailtu.

Algoritmien tilavaativuusanalyyseissä ei otettu huomioon algoritmin kutsujalta saamia muistivarauksia eikä algoritmin palauttamia muistivarauksia. Tosin jos se olisi otettu huomioon, niin ne eivät olisi vaikuttaneet algoritmien tilavaativuuksiin.

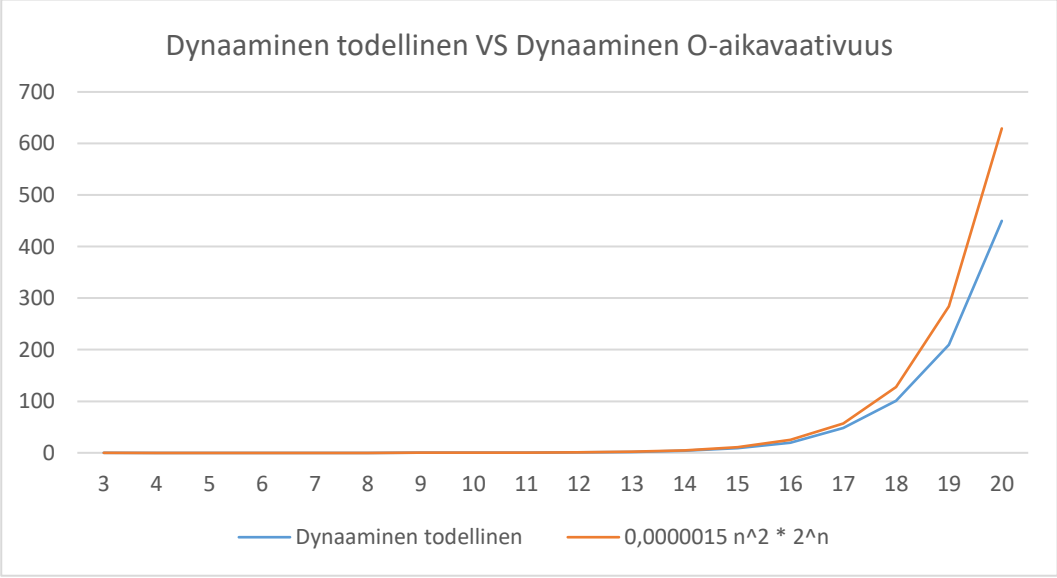
Suorituskyky- ja O-analyysivertailu

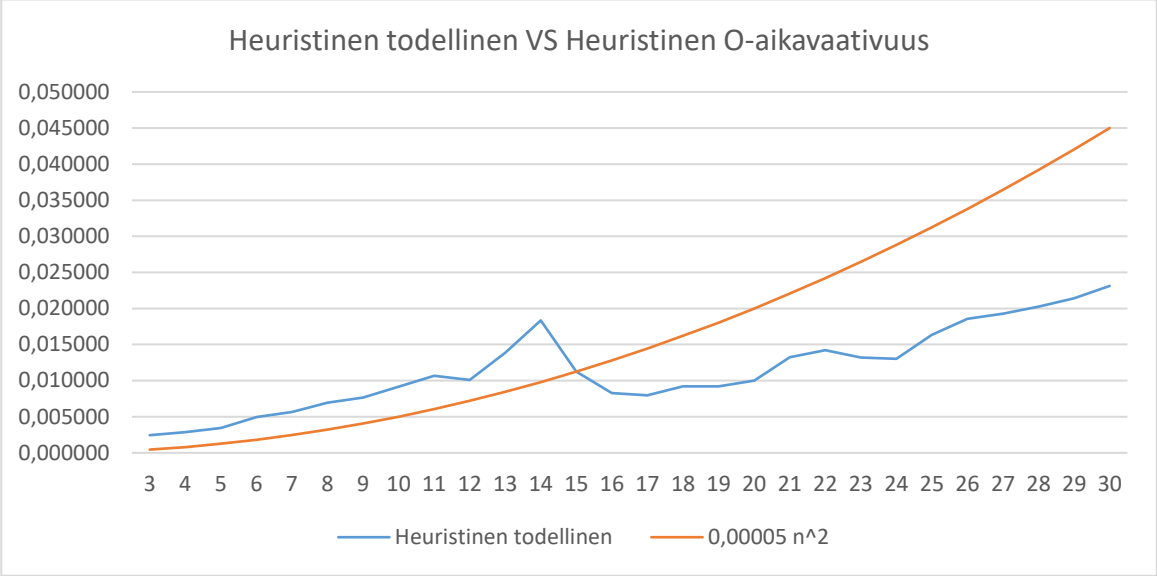
Testattu pääalgoritmien (BruteForce, Dynaaminen ja Heuristinen) ajallinen suorituskyky syöttämällä BruteForcelle verkkoja kokoväliltä 3-15 solmua, Dynaamiselle väliltä 3-20 solmua ja Heuristiselle väliltä 3-30 solmua. Arvottu jokaiseen verkon solmukokoluokkaan 10 erilaista verkkoa, sitten syötetty jokainen näistä verkoista 3 kertaa kullekkin pääalgoritmille ja sitten laskettu kyseisen algoritmin suoritusaika kyseiselle solmukokoluokalle näiden 30 suorituksen keskiarvona.

Alla esitettynä tulokset graafisesti. Jokaisessa kuvaajassa x-akselilla on aina verkon koko solmuina ja y-akselilla on aina kulunut aika millisekunteina (paitsi kahdessa kuvaajassa y-akselilla on prosentteja, mutta prosentit on merkitty erikseen).

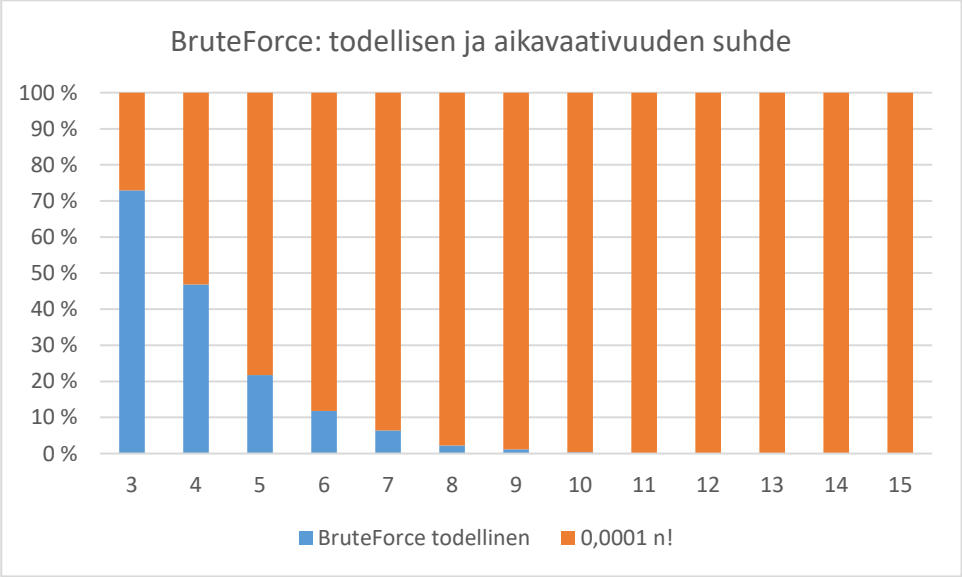


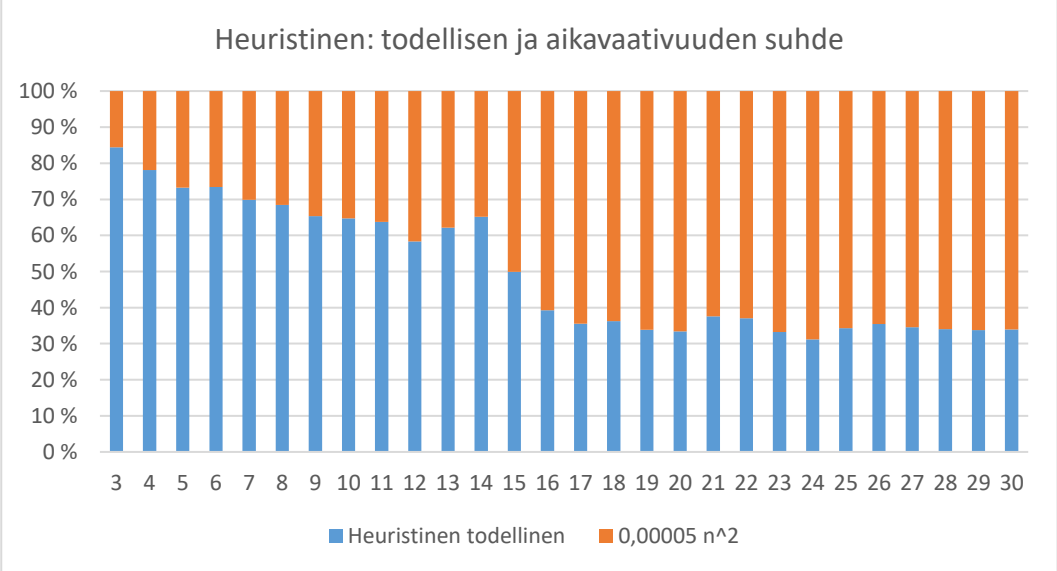
(BruteForcen kuvaaja näkyy paremmin yhteiskuvaajassa Dynaamisen ja Heuristisen kanssa näiden kuvaajien viimeisenä.)

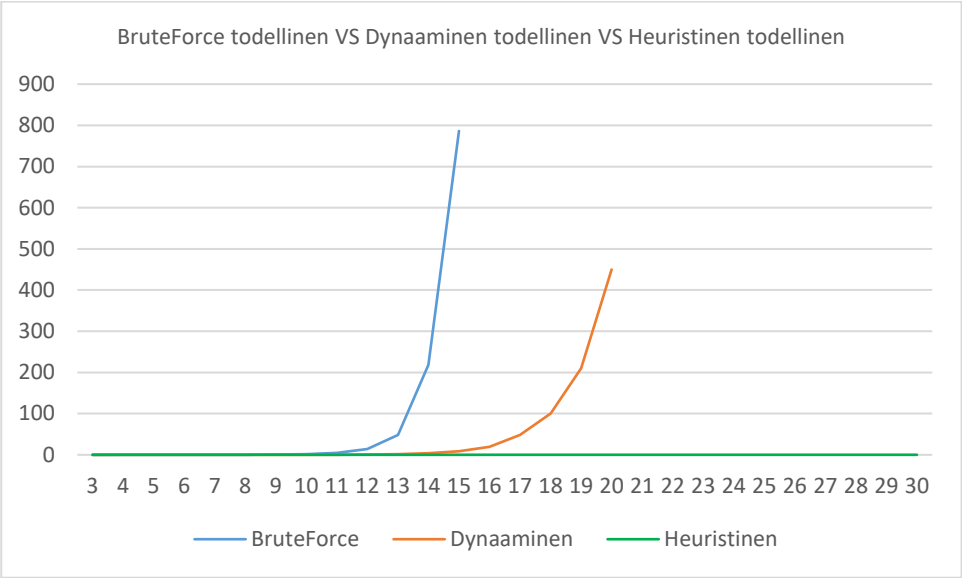




Kaksi seuraavaa kuvaajaa lisätty jotta BruteForcen ja Dynaamisen suhteet aikavaativuuksiinsa näkyy paremmin alkuosan suhteen joka hukkuu näkymättömiin koska loppuosassa mennään niin korkeisiin lukuihin.







Joten yhteenvetona voidaan sanoa, että algoritmit noudattavat O-aikavaativuuksiaan varsin tarkasti, paitsi BruteForcen vaatima aika lisääntyy dramaattisesti loivemmin verrattuna sen O-aikavaativuuteen. Tämä johtuu oletettavasti siitä, että algoritmi ei tutki reittejä joista tulee varmasti pitempiä kuin siihen mennessä löydetystä lyhimmästä reitistä. BruteForce on oikeastaan hieman huono nimi algoritmille, branch-and-bound olisi parempi nimi. Koska testit tehty verkoilla joissa solmujen etäisyydet arvottu väliltä 1-1000, niin tällöin tulee luultavasti hieman enemmän tilanteita joissa voidaan todeta että jostakin reitistä ei varmasti tule lyhintä, verrattuna verkkoon jossa solmujen etäisyydet olisivat kapeammalla alueella, esim. 900-1000.

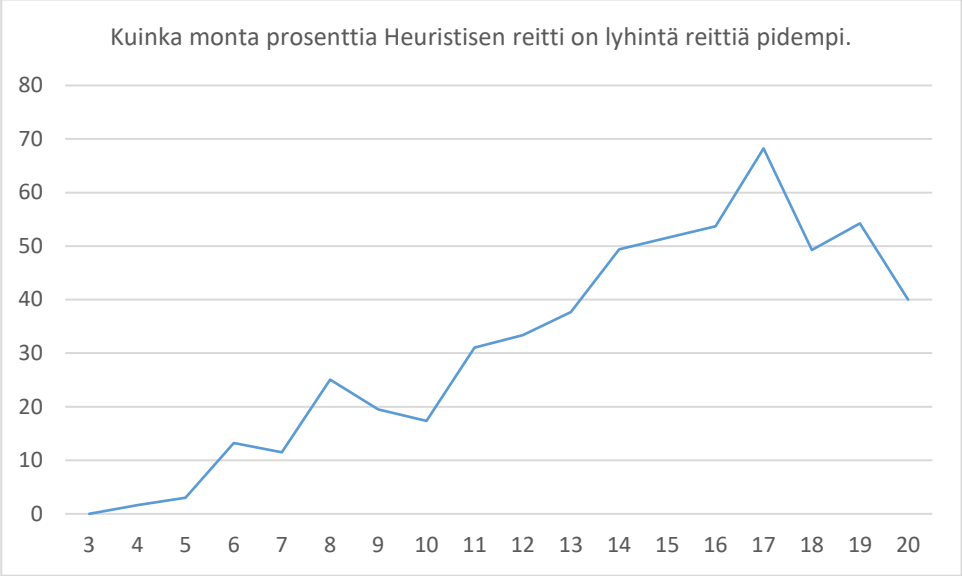
Nämä testit voi karkeasti toistaa (täysin samoja nanosekunteja testeistä ei tietenkään saa) MainTest-luokan aivan lopussa olevilla 3 metodilla, suorituskykyTestitBruteForce-metodilla, suorituskykyTestitDynaaminen-metodilla ja suorituskykyTestitHeuristinen-metodilla.

(Suorituskykytestit ovat tarkoituksella nimenomaan MainTest-luokan lopussa. Kun suorituskykytestit koitettu tehdä metodeilla jotka tiedoston keskivaiheilla, niin pienimpään verkkoon joka on ollut testausluupissa on tullut ihmeellinen lisä suoritusaikaan, ja tämä lisä on tasoittunut vasta kolmannen tai neljännen verkon kohdalla luupissa. Koitettu ongelmaan montaa eri ratkaisua, esim. sijoittaa testi omaan main-luokkaan, käyttää valmiiksi tehtyjä verkkoja eikä arvottuja verkkoja, suorittaa aina testi yhdellä verkolla kerrallaan. Ongelma ratkesi pääpiirteittäin vasta, kun testit siirretty aivan MainTest-luokan loppuun ja annettu edellä olevien testien olla aktiivisina. Tällöin suoritusajoissa oli havaittavissa enää todella pieni lisä, lähes olematon.)

Heuristisen reitinpituus verrattuna lyhimpään reittiin

Verrattu Heuristisen antamien reittien pituuksia lyhimpään reittiin kyseisessä verkossa (Laskettu lyhin reitti Dynaamisella). Arvottu jokaiseen verkon kokoluokkaan 10 verkkoa ja sitten annettu nämä verkot Heuristiselle (sekä Dynaamiselle) ja laskettu kuinka paljon pitempi Heuristisen antama reitti on prosentuaalisesti keskiarvoisesti verrattuna lyhimpään reittiin.

Alla esitetty tulos graafisesti. X-akselilla on verkon koko ja y-akselilla on Heuristisen reitin keskiarvoinen prosentuaalinen pituuslisä verrattuna lyhimpään reittiin.



Tämän testin voi toistaa MainTest-luokassa olevalla reitinPituudenEroDynaaminenVsHeuristinen-metodilla.

Työn mahdolliset puutteet ja parannusehdotukset

Dynaaminen algoritmi ei toimi asymmetrisillä verkoilla. Jos Dynaamisen algoritmin tekisi BruteForcen tyyliin mutta dynaamisella taulukoinnilla eikä Held-Karp tyyliin niin myös asymmetristen verkkojen pitäisi toimia.

Heuristisen reitinpituudet ovat todella paljon pitempiä verrattuna lyhimpään reittiin. Heuristisen voisi toteuttaa jollakin paremmalla algoritmilla.

Lähteet

[1] https://en.wikipedia.org/wiki/Held%E2%80%93Karp_algorithm#Complexity
[2] https://www.quora.com/Are-there-any-good-examples-of-the-Held-Karp-algorithm-in-C++-Hard-to-find-example-code-to-solve-the-traveling-salesman-problem-Everyone-wants-to-just-talk-about-theory-and-not-show-how-to-actually-do-it-What-is-the-big-secret