Eemeli Syynimaa

Optimointikirjaston suunnittelu robotisoidulle tuotantolinjalle

Insinööri (AMK)

Tieto- ja viestintätekniikka

Syksy 2017

KAJAANIN
AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Tiivistelmä

Tekijä: Syynimaa Eemeli

Työn nimi: Optimointikirjaston suunnittelu robotisoidulle tuotantolinjalle

Tutkintonimike: Insinööri (AMK), tieto- ja viestintätekniikka

Asiasanat: optimointi, tekoäly, algoritmi, ohjelmistokehitys

Työn tilaajana oli Raute Oyj:n Kajaanin yksikkö Mecano Business. Raute on suomalainen pörssiyhtiö, joka toimii puutoimialalla ja toimittaa teknologiaa ja palveluita maailmanlaajuisesti. Mecano toimittaa Rauten koneisiin konenäkö- ja analysointiteknologiaa.

Työn tavoitteena oli kehittää uusi optimointikirjasto eräälle Rauten robotisoidulle tuotantolinjalle. Optimoinnissa keskitytään robottien työtehtävien optimoimiseen. Työssä oli tarkoitus käyttää perustana vanhaa optimointikirjastoa ja toteuttaa aiemmin luodun simulaattorin avulla kehitettyjä optimointimenetelmiä.

Raportin alussa käsitellään algoritmien ja reitin optimoinnin teoriaa. Algoritmit ovat vaihe kerrallaan suoritettavia menetelmiä jonkin ongelman ratkaisemiseen tai tavoitteen saavuttamiseen. Reitin optimoinnissa etsitään optimaalista reittiä kaikista mahdollisista vaihtoehdoista ja se on tärkein optimointimenetelmä robottien työtehtävien optimoinnissa.

Työssä hyödynnettiin tuotantolinjan simulaattoria, jota voidaan pitää kehitetyn optimointikirjaston esiasteena. Toteutusvaiheessa käydään koko ohjelmistonkehityksen prosessi läpi: vaatimusmäärittely, suunnittelu, toteutus ja testaus. Kehitetyn optimoinnin työvaiheita ovat alustus, tehtävien lukeminen, tehtävien jakaminen, työjärjestyksen optimointi, tehtävälistan luonti sekä lopetus.

Kehitetty optimointikirjasto testattiin työn loppuvaiheilla. Testialustana käytettiin simulaattoria, koska oikeaa tuotantolinjaa ei ollut käytettävissä. Tulosten perusteella voidaan huomata, että kehitetty optimointikirjasto toimii. Optimointiparametrien käytön helpottaminen ja automatisointi toteutetaan myöhemmin jatkokehityksessä.

Lopputuloksena on täysin toimiva optimointikirjasto, jota voidaan ajaa niin simulaattorilla kuin oikealla tuotantolinjallakin, ja joka tarjoaa apua kapasiteetin mitoittamiseen ja tuotantolinjan kehittämiseen.

Abstract

Author: Syynimaa Eemeli

Title of the Publication: Optimization of Robotized Production Line

Degree Title: Bachelor of Engineering, Information Technology Engineering

Keywords: optimization, artificial intelligence, algorithm, software development

This thesis was ordered by Mecano Business, a unit of Raute Oyj located in Kajaani, Finland. Raute is a Finnish corporation that operates in the woods product industry and offers technology and services for global market.

The purpose of this thesis was to develop a new optimization library for one of Raute's automated production lines. The production line operates with multiple automated robots and the main objective in optimization is to optimize robot work schedule. This library was designed to be built upon the old optimization library and with the help of optimization methods developed with a previously built simulator.

The thesis details theory behind algorithms and route optimization. Algorithms are step-by-step methods for solving a specific problem or for reaching a specific goal. In route optimization, the goal is to find the most optimal route from all the possible ones. Route optimization is the most meaningful method in robot work schedule optimization.

A previously built simulator of the production line that was used in this thesis is the first step for this optimization library. The simulator and the library development process was thoroughly reviewed in the practice part of this thesis: requirements, planning, development and testing. The optimization library is split into following steps: initialization, task reading, task sharing, work schedule optimization, task list creation and deinitialization.

The developed optimization library was tested in the final development phase. The tests were carried on a simulator because no real production lines were available at the time. From the test results can be seen that the optimization library works. Optimization parameter usage improvements and automatization will be developed as a separate project in the future.

The result of this thesis is a fully operational optimization library that can be used with the simulator and with a real production line and that offers an elegant way to measure the production capacity and to improve the production line.

Sisällys

1 Johdanto

Optimointi on jokapäiväinen haaste teollisuudessa. Kaikilla aloilla etsitään jatkuvasti uusia tapoja kehittää tuotantomenetelmiä raaka-aineen ja kustannusten säästämiseksi sekä erityisesti kilpailukyvyn parantamiseksi.

Työn tilaajana toimii Raute Oyj:n Kajaanin yksikkö Mecano Business. Raute on suomalainen pörssiyhtiö, joka tarjoaa ratkaisuja puuteollisuuteen. Raute tarjoaa teknologiaa ja palveluita esimerkiksi vaneriteollisuuteen, jonka markkinajohtaja Raute on 15–20 prosentin markkinaosuudella. Rauten päätoimipiste sijaitsee Lahden Nastolassa ja Mecanon tuotantoyksikkö Kajaanissa. Muut omat tuotantoyksiköt löytyvät Kanadasta, Kiinasta ja Yhdysvalloista. Rauten tarjoama teknologia kattaa kaikki tuotantoprosessiin kuuluvat koneet ja laitteet sekä palvelut näiden ylläpitoon. [1.]

Mecano kehittää tuotantokappaleiden analysointia, mikä on oleellinen komponentti useissa Rauten tarjoamissa koneissa ja tuotantolinjoissa. Analysoinnissa haetaan tuotantokappaleiden ominaisuudet, joiden perusteella muu laitteisto käsittelee tuotantokappaleita oikealla tavalla. Se ei kuitenkaan rajoitu pelkästään tiedon keräämiseen, vaan se sisältää myös analysoidun datan käsittelyä. Tämä työ käsittelee yhtä analysoinnin työvaihetta, optimointia.

Rautella on eräs tuotantolinja, jonka työkuormaa halutaan optimoida. Tuotantolinjalla toimii yksi tai useampia robotteja, jotka automaattisesti suorittavat niille määritettyjä tehtäviä. Linjalla kulkee tuotantokappaleita, ja kun ne saapuvat jonkin robotin työskentelyalueelle, suorittaa robotti sille määritellyt tehtävät kyseiselle tuotantokappaleelle. Kun tehtävät on suoritettu, siirtyy tuotantokappale eteenpäin seuraavan robotin työskentelyalueelle. Optimitilanteessa linja toimii koko ajan täydellä teholla. Jos jollain robotilla on paljon tehtäviä, linjaa voidaan hidastaa tai se voidaan pysäyttää kokonaan.

Työn tavoitteena on robottien tehtävien optimointi siten, että linja joutuisi hidastamaan mahdollisimman vähän. Optimoinnista voidaan eriyttää kaksi merkittävää haastetta: tehtävien jakaminen roboteille ja robotin työjärjestyksen optimointi. Tässä raportissa keskitytään ensisijaisesti työjärjestyksen optimointiin.

Robotin tehtävänä on käydä jokaisella sille määritellyllä tehtäväpisteellä ja suorittaa määritetty tehtävä. Tehtävien suoritusjärjestyksellä ei ole tuotteen laadun kannalta merkitystä, joten sitä voidaan vapaasti optimoida: on hyödyllistä välttää tarpeettomat rakenteita kuormittavat kiihdytykset tai muut epäsuotuisat liikkeet. Ensisijaisena tavoitteena onkin löytää

mahdollisimman lyhyt reitti, joka kertaalleen käy jokaisessa tehtäväpisteessä. Täten voidaan todeta, että kyseessä on reitinhakuongelma.

Työn esiasteena voidaan pitää samaan tuotantolinjaan perustuvaa simulaattoria. Sen kehityksen yksi päätavoitteista oli uusien optimointimenetelmien ja -algoritmien tutkiminen, ja näitä havaintoja käytetään hyväksi tässä työssä. Simulaattori toimii myös testiympäristönä työn toteutusvaiheessa.

Työ toteutetaan luomalla uusi, itsenäisesti toimiva optimointikirjasto osaksi kohdeyrityksen nykyistä ohjelmistoa. Kun tuotantolinjan mittaustuloksista määritetään uusia tehtäviä, lähetetään ne optimointikirjastolle yksinkertaisen käyttöliittymän kautta, jolta ne palautuvat optimoituna takaisin lähetettäväksi edelleen roboteille.

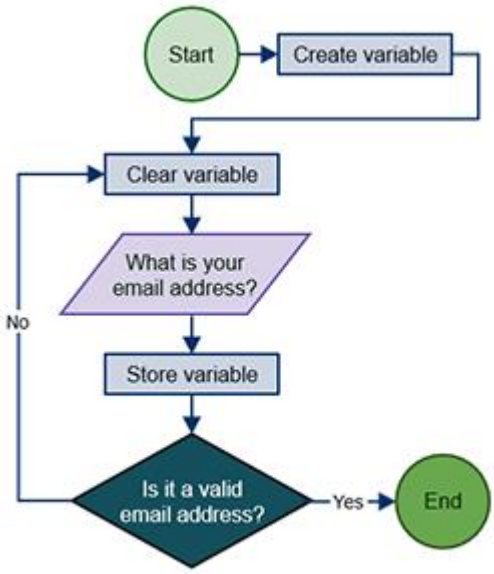
Seuraavassa osassa tutustutaan algoritmien teoriaan, jonka jälkeen paneudutaan työn aiheena olevan optimointitehtävän suurimpaan haasteeseen, eli reitin optimointiin. Samalla esitetään työssä käytettyjä algoritmeja. Seuraavaksi esitetään simulaattori ja sen hyödyntämismahdollisuudet. Käytännön osuudessa esitetään työn toteutus: optimointikirjasto. Lopuksi tutkitaan simulaattorin avulla optimoinnin lopputuloksia ja esitetään jatkokehitysideoita sekä päätetään raportti pohdintaan ja yhteenvetoon.

2 Algoritmeista yleisesti

Ohjelmistonkehityksessä pyritään ratkomaan ongelmia, joiden asettelu on yleensä hyvin selkeä: tietyllä alkusyötteellä pitäisi saada haluttu lopputulos. Ongelmien haaste tuleekin niiden ratkaisemisesta: millä menetelmällä haluttuun lopputulokseen voidaan päästä? Näitä menetelmiä kutsutaan algoritmeiksi.

2.1 Algoritmien toiminta

Algoritmi on vaihe kerrallaan suoritettava menetelmä jonkin ongelman ratkaisemiseen tai tavoitteen saavuttamiseen [2]. Eri suoritusvaiheet pyritään esittämään mahdollisimman yksityiskohtaisesti, jotta algoritmin käyttö olisi helppoa. Kuvassa 1 on esimerkki yksinkertaisesta algoritmista.



Kuva 1. Esimerkki yksinkertaisesta algoritmista [3]

Algoritmi terminä on perinteisesti mielletty osaksi matematiikkaa ja ohjelmistonkehitystä, mutta nykyään sitä käytetään yleisesti myös muissa yhteyksissä. Voidaan esimerkiksi sanoa, että Rubikin kuution ratkaisu tai pizzan valmistusohje ovat algoritmeja [2].

Koska algoritmit kuvaavat ongelman ratkaisun yleisellä tasolla, eivät ne sellaisenaan käy ratkaisuksi ohjelmointiongelmiin. Algoritmin toteutus pitää aina erikseen ohjelmoida haluttuun ympäristöön. Tästä johtuen algoritmit ovat yleensä riippumattomia käyttöjärjestelmästä tai ohjelmointikielestä: vain niiden toteutus vaihtelee. [4, s. 4.]

2.2 Algoritmien tehokkuus ja vertailu

Yleensä samaan ongelmaan löytyy useita käyttökelpoisia ratkaisuja, jotka vaihtelevat suorituskyvyssä ja muistinkäytössä. Pienille ongelmille kelpaa ratkaisuksi käytännössä mikä tahansa algoritmi, mutta isoille ongelmille oikean algoritmin valinnalla voi olla lopputuloksen kannalta suuri merkitys. Ongelmissa, joissa on miljoonia eri objekteja, voidaan hyvin suunnitellulla algoritmilla saada miljoonakertainen nopeusetu huonosti suunniteltuun algoritmiin nähden. Algoritmien suunnittelu onkin hyvin tärkeää, sillä laskenta- tai muistikapasiteetin lisääminen on aina kallista ja niiden tuoma nopeusetu jää todennäköisesti huomattavasti pienemmäksi. [4, s. 5.]

Algoritmeja vertailtaessa on vakiintuneeksi käytännöksi muodostunut asymptoottinen notaatio (*Big-Oh-notaatio* tai *O-notaatio*). Sen keskiössä on *N*, joka kuvaa jonkin algoritmin suoritusaikaan eniten vaikuttavaa parametria ja on usein suoraan verrannollinen käsiteltävän tietorakenteen kokoon [4, s. 36–37].

Asymptoottisessa notaatiossa muut parametrit hylätään merkityksettöminä. Esimerkiksi kaavassa $N^2 + c$ termillä *c* ei ole suoritusajan kannalta mitään merkitystä, sillä N^2:n laskemiseen kuluu huomattavasti enemmän aikaa. Voidaankin todeta, että algoritmin suoritusaika on yhtä kuin $O(N^2)$ eli suoritusaika kasvaa neliöllisesti syötteiden lukumäärän funktiona. Notaation ei ole tarkoitus kuvata algoritmien suoritusaikaa mahdollisimman tarkasti, vaan tehdä niiden analysoinnista yksinkertaisempaa [4, s. 44].

Algoritmien analysoinnissa on kaksi merkittävää ja toisistaan eroavaa näkökulmaa: huonoimman tilanteen analysointi ja keskimääräisen tilanteen analysointi. [4, s. 60.]

Huonoimman tilanteen analysoinnissa on tavoitteena löytää ja tehdä lupaus algoritmin huonoimmasta mahdollisesta suoritusajasta, mikä on yksi algoritmien analysoinnin perustavoitteista. Tässä näkökulmassa on kuitenkin muutamia ongelmia. Algoritmin keskimääräinen suoritusaika saattaa olla hyvin kaukana huonoimmasta suoritusajasta, mikä antaa vääristyneen kuvan algoritmin keskimääräisestä suoritusajasta. Lisäksi algoritmin toteu-

tus saattaa olla niin monimutkainen, että vaikka huonoin suoritusaika on matala, niin keskimääräinen suoritusaika saattaa olla jotain toista, huonoimmalta suoritusajaltaan heikompaa algoritmia heikompi. [4, s. 60.]

Keskimääräisen tilanteen analysoinnissa puolestaan haetaan algoritmin suoritusajan keskiarvoa, jolla voidaan tehdä ennustuksia toteutuksen suoritusajoista. Tästäkin näkökulmasta löytyy omat haasteensa. Esimerkiksi keskimääräisen suoritusajan arviointi voi olla matemaattisesti hyvin hankalaa ja vaikeasti perusteltavissa. [4, s. 61.]

Algoritmien tehokkuuden määrittelemiseen käytetään *N*-parametria, mutta missä vaiheessa voidaan sanoa, että algoritmi on *tehokas*? Yksi yleisesti käytössä oleva menetelmä on määritellä algoritmi *tehokkaaksi*, jos algoritmin huonoin suoritusaika on polynomisesti kasvava. Eksponentiaalisesti kasvava algoritmi saattaa olla yksinkertaisissa tapauksissa nopeampi, mutta on aina olemassa *N*-parametri, jonka ylittyessä eksponentiaalisen algoritmin suoritusaika ylittää polynomisen algoritmin suoritusajan. [5, s. 43–44.]

Usein tulee vastaan ongelmia, joiden tunnetut ratkaisualgoritmit eivät ole tehokkaita: optimaalisen ratkaisun laskemiseen kuluu todella paljon aikaa eikä niiden käyttö ole aina mahdollista. Näissä tapauksissa on perusteltua käyttää algoritmeja, jotka eivät välttämättä löydä optimaalista ratkaisua, mutta jotka kuitenkin löytävät lähellä olevan ratkaisun. Näitä algoritmeja kutsutaan approksimointialgoritmeiksi [6].

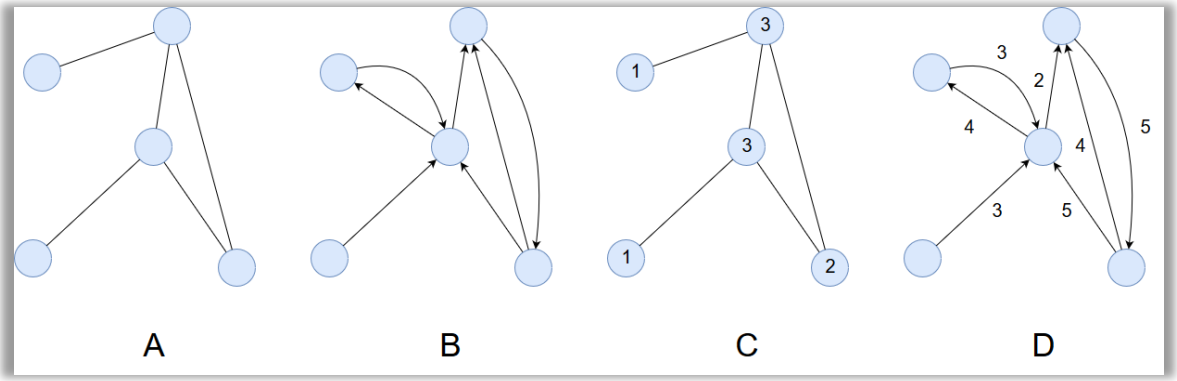
3 Reitin optimointi

Reitin optimoinnissa etsitään optimaalista reittiä kaikista mahdollisista vaihtoehdoista. Optimaalinen reitti on kuitenkin yleisenä määritelmänä hämärä, sillä se riippuu kyseessä olevasta ongelmasta.

Reitin optimoinnin kanssa käsi kädessä kulkee reitinhaku. Molemmilla on hyvin paljon samoja ominaisuuksia: reitinhaussa pääasiallisena tavoitteena on löytää reitti, mutta sen on myös hyvä olla optimaalinen. Reitinhakua käytetään erityisesti tekoäly- ja pelisovelluksissa [7, s. 716].

3.1 Verkkojen teoriaa

Verkko (engl. *graph*) on matemaattinen rakenne, jota voidaan käyttää hyödyksi reitin optimoinnin ja reitinhaun ongelmissa. Käsiteltävästä ongelmasta voidaan luoda yksinkertainen verkko, jossa on esiteltynä vain ongelman ratkaisun kannalta oleellinen informaatio. Verkko muodostuu solmuista (engl. *vertex*) ja näitä yhdistävistä kaarista (engl. *arc*) tai suunnatuista kaarista eli nuolista (engl. *arrow*) [8, s. 1]. Kuvassa 2 on esitetty erilaisia verkkoja.



Kuva 2. Suuntaamaton verkko, suunnattu verkko, verkon solmujen asteet ja painotettu verkko

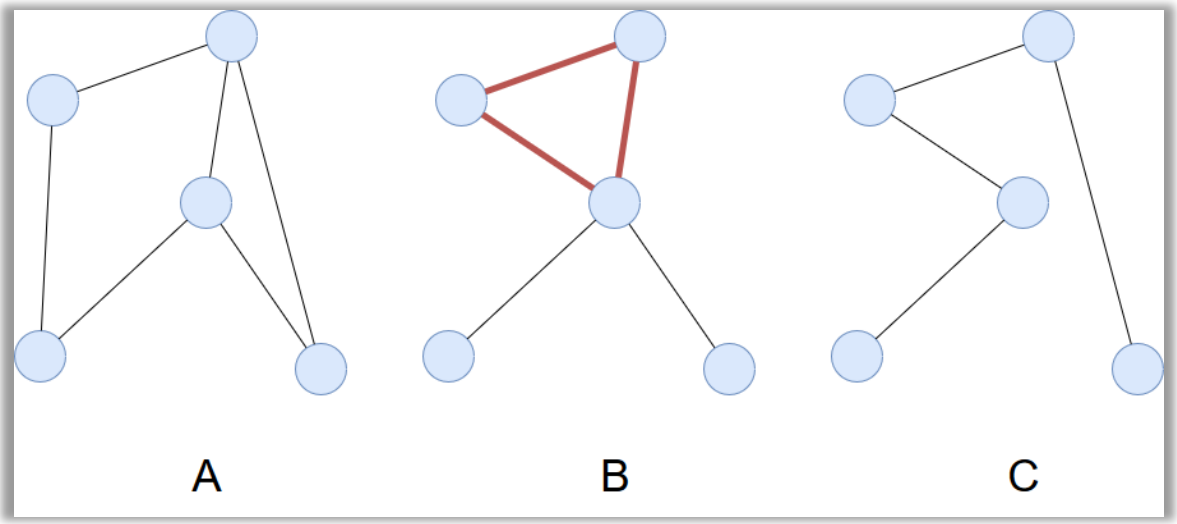
Kuvan 2 verkko A on suuntaamaton verkko. Sen solmuja yhdistävät kaaret, jotka mahdollistavat liikkumisen molempiin suuntiin. Verkko B puolestaan on suunnattu verkko, jonka solmuja yhdistävät nuolet. Nuolta pitkin voidaan liikkua vain sen osoittamaan suuntaan.

Kahta solmua voi yhdistää useampi nuoli, mikä saattaa mahdollistaa edestakaisen liikkumisen solmujen välillä. [8, s. 1.]

Jokaisella verkon solmulla on vähintään yksi asteluku. Suuntaamattomalla verkolla solmulla on yksi asteluku, joka kuvaa solmuun yhteydessä olevien kaarien määrää [8, s. 3]. Suunnatuilla verkoilla joka solmulle lasketaan erikseen siihen saapuvien ja siitä lähtevien nuolten lukumäärät [8, s. 10]. Kuvassa 2 verkossa C on esitetty suuntaamattoman verkon solmujen asteluvut.

Lähtökohtaisesti solmuja yhdistävät kaaret ja nuolet ovat samanarvoisia: niitä käsitellään yhtä pitkinä ja niiden pituus vaihtelee ainoastaan havainnoinnin helpottamiseksi. Kuvan 2 verkko D on painotettu verkko, jonka nuolille on annettu jokin painokerroin [8, s. 13]. Painokertoimella voidaan kuvata esimerkiksi solmujen välisiä etäisyyksiä, ja niitä voidaan käyttää sekä suunnatuissa että suuntaamattomissa verkoissa [8, s. 13].

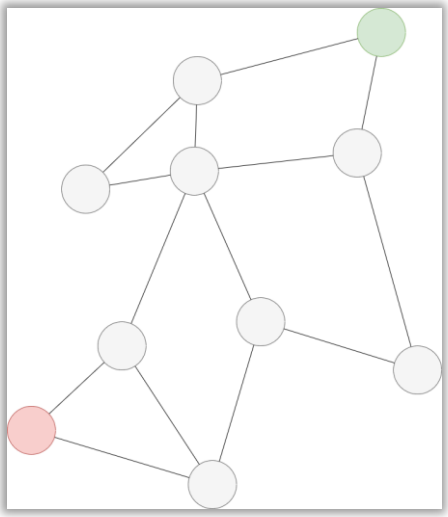
Verkosta voidaan muodosta puu, jos se on yhtenäinen eikä siinä ole suljettuja ketjuja solmujen välillä [8, s. 14]. Verkko on yhtenäinen (engl. *connected*), jos sen jokaisen solmuparin välillä on yhteys [8, s. 7]. Suljettu (engl. *cycle*) ketju muodostuu, jos mistä tahansa solmusta lähtevä ketju palaa takaisin lähtösolmuun [8, s. 6]. Kuvassa 3 on havainnollistettu yhtenäinen verkko A, suljettu ketju B ja puu C.



Kuva 3. Yhtenäinen verkko, suljettu ketju ja puu

3.2 Reitinhakuongelmia

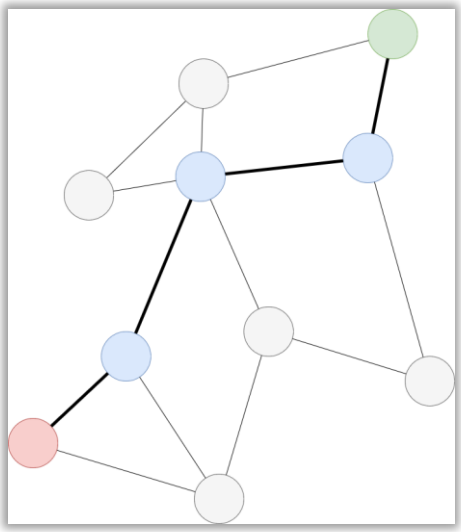
Reitinhakuun liittyy useita erilaisia ongelmia. Useimmilla on kuitenkin samanlainen lähtökohta: jokin verkko, joka toimii matemaattisena rakenteena reitin hakualueesta ja johon on määritelty aloitus- ja lopetussolmut. Kuvassa 4 on esimerkki reitinhaun kohteena olevasta verkosta, johon on aloituspiste merkitty vihreällä ja lopetuspiste punaisella.



Kuva 4. Reitinhaun kohteena oleva verkko

3.2.1 Lyhyimmän reitin ongelma

Lyhyimmän reitin haku on varmasti yksi vanhimpia ja samalla yksi helpoimmin omaksuttavia ongelmia. Voidaankin kuvitella, että jo hyvin primitiivisissä yhteiskunnissa on ymmärretty lyhyimpien reittien merkitys esimerkiksi ravinnon luokse [9, s. 155]. Se on konseptina yksinkertainen: kahden pisteen välillä on aina jokin etäisyys, joka on muita lyhyempi, ja joka usein halutaan löytää. Kuvassa 5 on esimerkki lyhyimmän reitin ongelmasta.



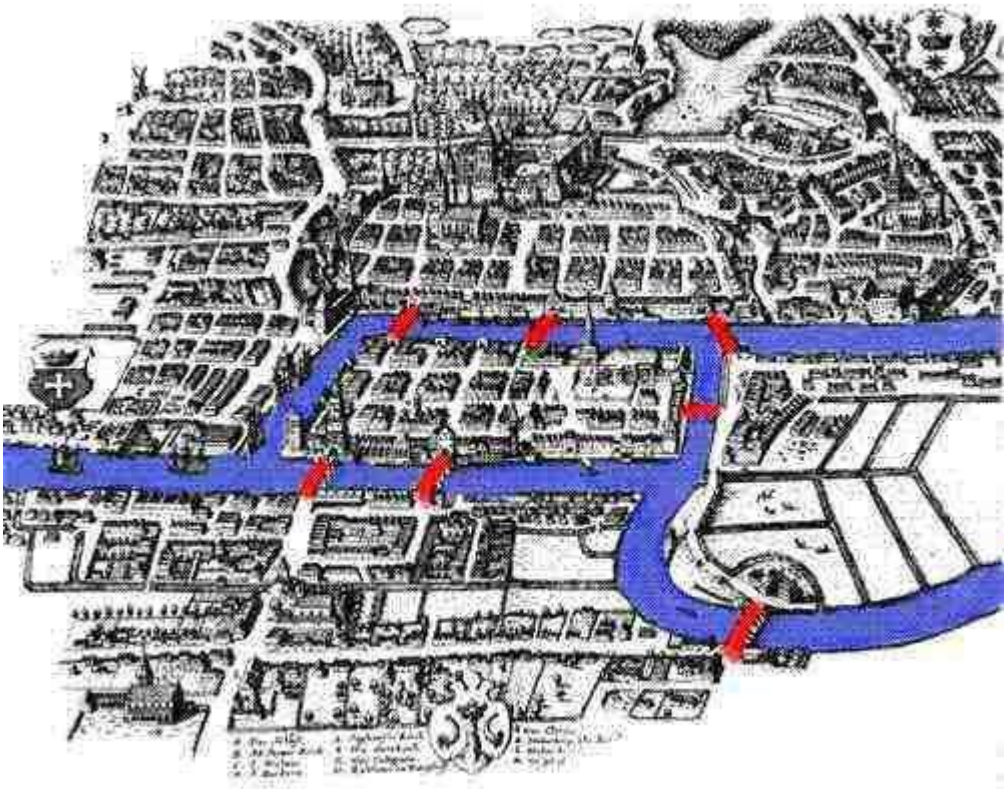
Kuva 5. Esimerkki lyhyimmän reitin ongelmasta

Vaikka lyhyimmän reitin haku on ongelmana hyvin yleinen, on sitä muihin optimointiongelmiin verrattuna alettu tutkimaan verrattain myöhäisessä vaiheessa. Tämä saattaa juontaa ongelman yksinkertaisuudesta. Tähän myös viittaa se, että useampi tutkija kehitti itsenäisesti hyvin samankaltaisia ratkaisuja. [9, s. 155.]

Lyhyimmän reitin hakuun löytyy useita ratkaisuja. Esimerkkejä hyvin tunnetuista algoritmeista ovat leveyshaku (engl. *breadth-first search*), syvyyshaku (engl. *depth-first search*), Djikstran algoritmi sekä A*-algoritmi (A-tähti tai engl. A-star) [10, s. 756].

3.2.2 Königsbergin siltaongelma

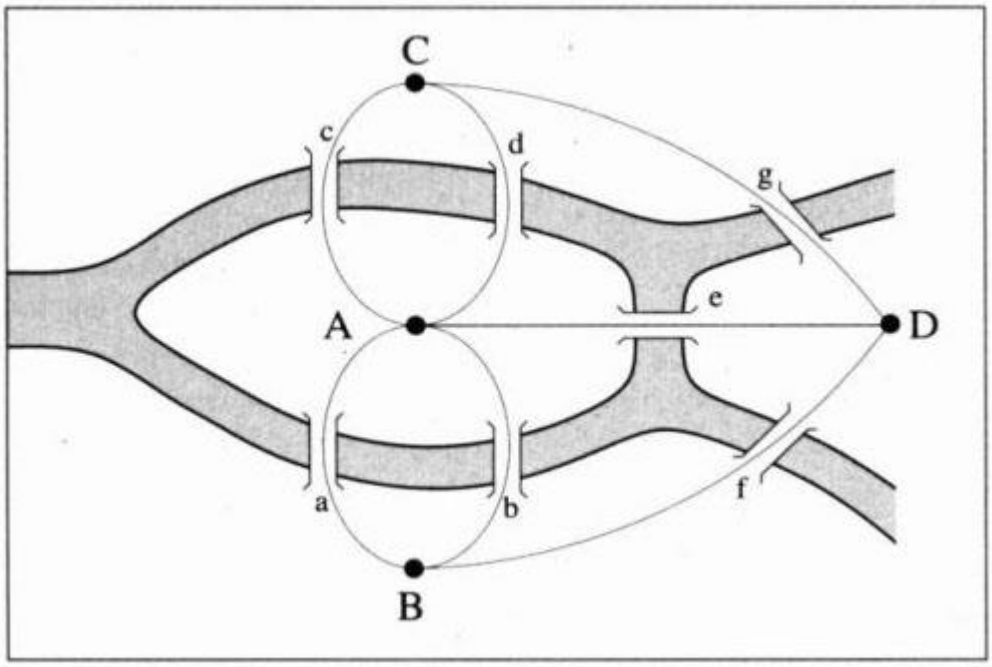
Königsbergin siltaongelma sijoittuu Königsbergiin (nyk. Kaliningrad). Königsberg sijaitsee Pregoljajoen molemmilla rannoilla ja niiden väliin jäävillä kahdella saarella. Saaria ja rantoja yhdistää seitsemän siltaa kuvan 6 mukaisesti. [11, s. 199–200].



Kuva 6. Königsbgergin siltaongelma [12]

Ongelmana on löytää reitti, jota kulkemalla voitaisiin ylittää jokainen silta täsmälleen yhden kerran ja päätyä takaisin lähtöpisteeseen. Sveitsiläinen matemaatikko Leonhard Euler todisti vuonna 1735, että ongelmaan ei ole ratkaisua, ja loi samalla pohjan verkkoteorialle. [13, s. 35–36.]

Euler loi ongelmasta kuvan 7 mukaisen verkon. Verkossa isoilla kirjaimilla merkityt maa-alueet ovat solmuja ja niitä yhdistävät pienillä kirjaimilla merkityt sillat ovat solmuja yhdistäviä kaaria. Euler todisti verkon avulla, että ongelmaan ei voi olla ratkaisua, koska paritonasteisten solmujen määrä ei ole nolla tai kaksi. Parittomien solmujen teoria pohjautuu siihen, että jos johonkin solmuun päädytään yhtä kaarta pitkin, niin vastaavasti siitä solmusta pitää päästä pois jotain toista kaarta pitkin. Muuten samaa kaarta käytettäisiin kahdesti, mikä ei sovi ongelman asetelmaan. Jos taas parittomien solmujen lukumäärä on kaksi, niin parittomat solmut toimivat luonnollisina aloitus- ja lopetuspisteinä. [11, s. 199–201.]



Kuva 7. Königsbergin ongelmasta luotu verkko [14]

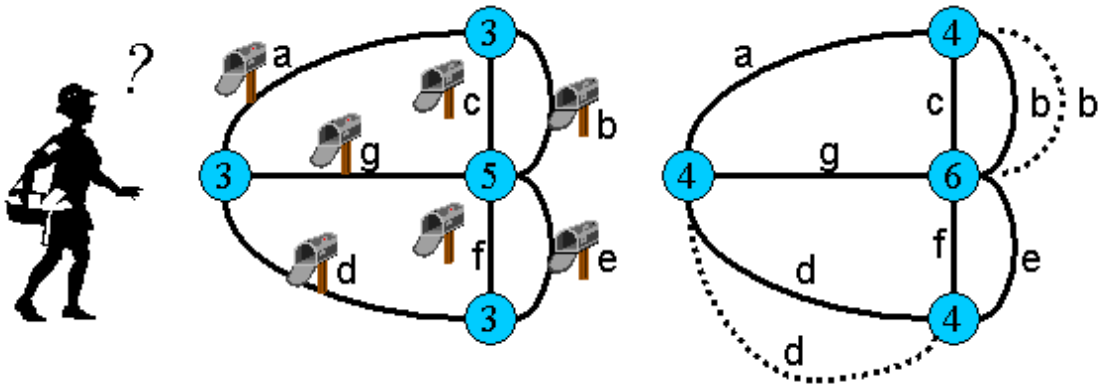
Tämän ongelman havaintojen pohjalta Euler toi tunnetuksi käsitteet Eulerin polku (engl. *Eulerian path*) ja Eulerin kierros (engl. *Eulerian circuit*). Eulerin kierros on olemassa, jos yhtenäisen, suuntaamattoman verkon paritonasteisten solmujen määrä on nolla tai kaksi, ja Eulerin polku on olemassa, jos Eulerin kierroksella on kaksi paritonta solmua jolloin ne muodostavat aloitus- ja lopetuspisteet. Jos verkko toteuttaa Eulerin kierroksen tai polun, voidaan se etsiä esimerkiksi Fleuryn algoritmilla [15].

3.2.3 Kiinalaisen postimiehen ongelma

Kiinalainen matemaatikko Kwan Mei-Ko loi 1960-luvun alussa mukaelman Königsbergin siltaongelmasta, kiinalaisen postimiehen ongelman. Kiinalaisen postimiehen tehtävänä on jakaa posti jokaiselle kaupungin kadulle ja ongelmana löytää reitti, jossa jokainen katu kuljetaan läpi vähintään kerran ja kuljettu matka on mahdollisimman lyhyt. [16.]

Myös kiinalaisen postimiehen ongelmasta voidaan luoda verkko: siinä risteykset ovat solmuja ja kadut niitä yhdistäviä kaaria. Ongelmassa on sama perusajatus kuin Königsbergin siltaongelmassa, mutta kiinalainen postimies saa kulkea saman kadun läpi useammin kuin kerran. Joissain tapauksissa kiinalaisen postimiehen verkosta kuitenkin löytyy Eulerin kierros, joka tarkoittaa automaattisesti lyhyintä mahdollista reittiä. [16.]

Kuvassa 8 on esimerkki kiinalaisen postimiehen ongelmasta.

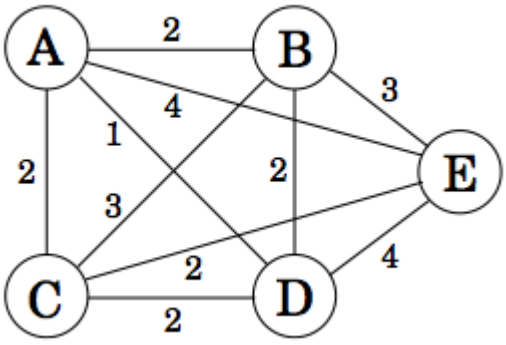


Kuva 8. Esimerkki kiinalaisen postimiehen ongelmasta [17]

3.2.4 Kauppamatkustajan ongelma

Eräällä kauppamatkustajalla on lista kohteita, ja hän tietää jokaisen kohteen etäisyyden muihin kohteisiin. Kauppamatkustajan pitäisi aloittaa käyntikierros kotoaan, ja hänen olisi käytävä jokaisessa listan kohteessa kerran ja palattava sitten takaisin kotiinsa. Mutta kauppamiehellä on ongelma: missä järjestyksessä kohteet on kierrettävä, että kuljettu matka olisi mahdollisimman lyhyt? [5, s. 1.]

Kuvassa 9 on esimerkki kauppamatkustajan ongelmasta.



Kuva 9. Esimerkki kauppamatkustajan ongelmasta [18]

Kauppamatkustajan ongelma on asettelultaan hyvin lähellä kiinalaisen postimiehen ongelmaa. Näissä kahdessa on kuitenkin se ero, että kauppamiehen ei tarvitse kulkea jokai-

sen kaaren läpi, toisin kuin kiinalaisen postimiehen. Kauppamiehelle riittää, että jokaisessa solmussa on käyty kertaalleen, välittämättä siitä, jäikö jokin yksittäinen kaari kulkematta.

Yksinkertaisesta asettelusta huolimatta kauppamatkustajan ongelmaan on vaikea löytää täydellistä ratkaisua. Jokaisen reittivaihtoehdon läpikäyntiin kuluu todella paljon aikaa, mikä on harvoin mielekästä. Useita approksimointialgoritmeja on ehdotettu, mutta nekään eivät varmuudella löydä lyhyintä reittiä, vaikka hyviä tuloksia niilläkin saadaan. [5, s. 17.]

Kauppamatkustajan ongelma ei kuitenkaan rajoitu pelkästään tähän yksinkertaiseen asetteluun. Ongelma on helposti yhdistettävissä eri tilanteisiin, ja siihen soveltuvat ratkaisut ovat käyttökelpoisia esimerkiksi logistiikassa, mikrosirujen valmistuksessa ja jopa avaruusteknologiassa [19]. Lisäksi tämän työn reitin optimointitehtävä vastaa täydellisesti kauppamatkustajan ongelmaa, joten myös siihen voidaan soveltaa eräitä tunnettuja approksimointialgoritmeja: Christofides-algoritmia ja 2-opt-algoritmia. Molemmat algoritmit esitetään seuraavassa luvussa.

3.3 Reitinhakualgoritmit

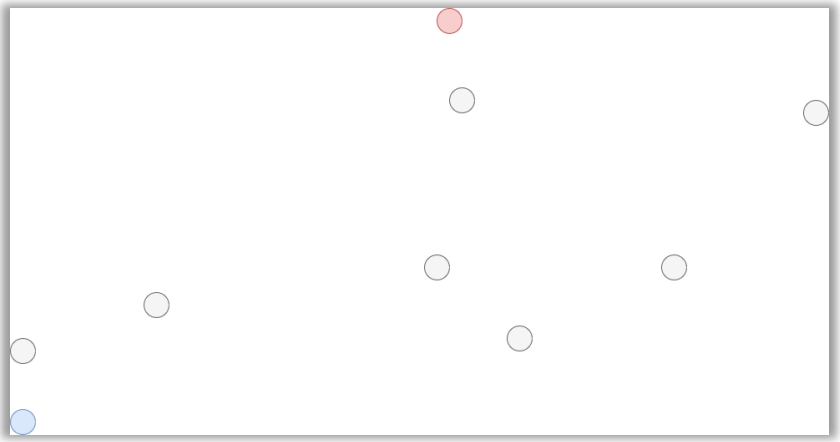
Seuraavissa alikappaleissa tutustutaan kahteen tunnettuun approksimointialgoritmiin: Christofides-algoritmiin ja 2-opt-algoritmiin.

3.3.1 Christofides-algoritmi

Christofides-algoritmi on Nicos Christofiden vuonna 1976 julkaisema approksimointialgoritmi kauppamatkustajan ongelmaan. Sen suoritusaika on verrannollinen syötteiden lukumäärän kolmanteen potenssiin eli $O(n^3)$, ja se löytää varmuudella reitin, joka on pituudeltaan korkeintaan 3/2-kertainen optimaaliseen reittiin verrattuna. Reitin pituus oli aikoinaan suuri harppaus, sillä aikaisemmat algoritmit löysivät huonoimmillaan kaksinkertaisia reittipituuksia optimaaliseen reittiin verrattuna. [20, s 1.]

Christofides-algoritmin syötteenä toimii suuntaamaton verkko. Verkossa on esitetty kauppamatkustajan vierailtavat kaupungit solmuina, ja niitä yhdistävät kaaret kuvaavat kaupunkien välisiä etäisyyksiä. Kaikki kaupunkien väliset etäisyydet huomioidaan. Kuvassa 10 on esitetty alkusyötteenä toimiva verkko, jossa sininen solmu on lähtöpiste ja punainen

solmu on loppupiste. Solmujen väliset kaaret on jätetty piirtämättä niiden suuren määrän takia.

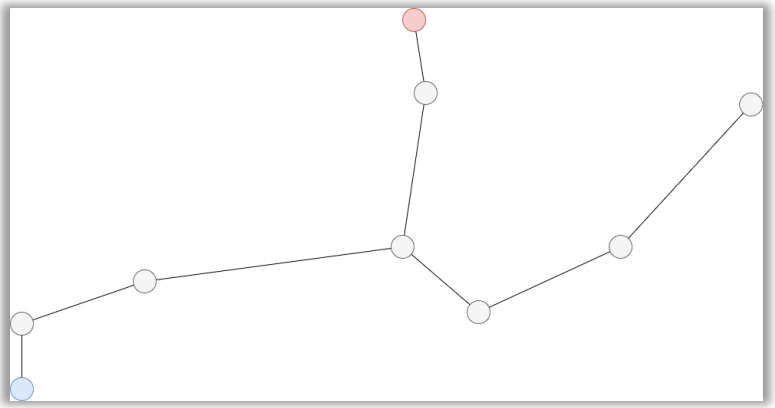


Kuva 10. Alkusyöte

Christofides-algoritmi on itsessään kokoelma muita tunnettuja algoritmeja ja ongelmia, ja se voidaan luontevasti jakaa useampaan eri työvaiheeseen. Ensimmäisessä vaiheessa etsitään alkusyötteenä toimivasta verkosta pienin virittävä puu. Toisessa vaiheessa suoritetaan paritonasteisten solmujen täydellinen minimisovitus ja yhdistetään se puuhun. Kolmannessa vaiheessa etsitään puusta Eulerin polku. Neljännessä vaiheessa Eulerin polusta luodaan Hamiltonin polku, joka toimii myös Christofides-algoritmin lopputuloksena. Työvaiheet esitetään yksityiskohtaisemmin omissa aliluvuissaan.

Vaihe 1: Pienimmän virittävän puun etsiminen

Virittävä puu on aliverkko, joka sisältää verkon jokaisen solmun mutta mahdollisimman vähän kaaria [8, s. 16]. Yhdellä verkolla voi olla useita virittäviä puita [21], mutta tässä vaiheessa on tavoitteena löytää puu, jonka paino, eli kaarien yhteenlaskettu pituus, on mahdollisimman pieni [22, s. 487]. Kuvassa 11 on esimerkki pienimmästä virittävästä puusta.

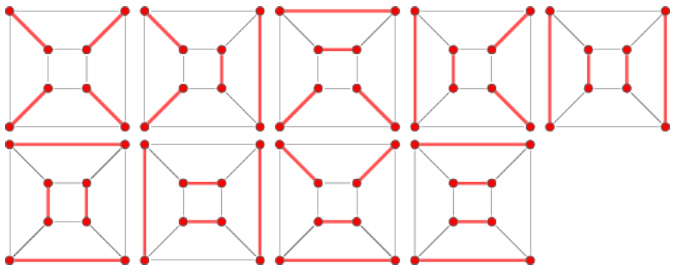


Kuva 11. Pienin virittävä puu

Pienimmän virittävän puun etsimiseen voidaan käyttää esimerkiksi Primin algoritmia. Se soveltuu myös suurimman virittävän puun, jonka paino on mahdollisimman suuri, etsimiseen [23, s. 286].

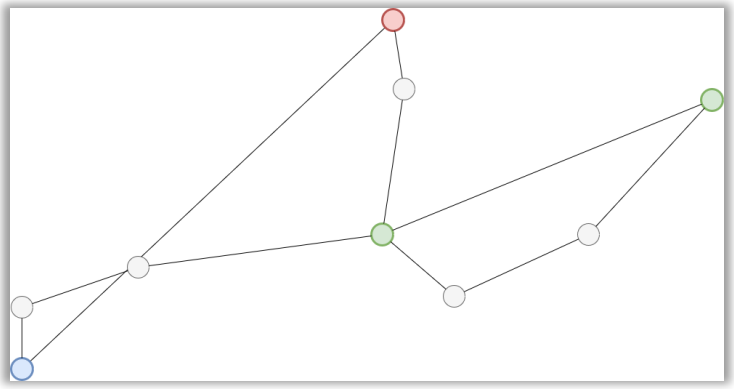
Vaihe 2: Paritonasteisten solmujen täydellinen minimisovitus

Seuraavaksi haetaan puusta kaikki paritonasteiset solmut (engl. *odd degree*). Jokaiselle solmulle etsitään pari täydellisellä minimisovituksella (engl. *minimum weight perfect matching*). Jonkin verkon täydellisessä sovituksessa jokaiseen solmuun on yhteydessä ainoastaan yksi kaari, ja täydellisessä minimisovituksessa näiden kaarien paino on pienin mahdollinen [24, s. 138]. Kuvassa 12 on erään verkon kaikki täydelliset sovitukset, jossa punaiset ympyrät ovat solmuja ja jotka muodostavat niihin yhdistettyjen punaisten kaarien kanssa täydellisen sovituksen.



Kuva 12. Erään verkon kaikki täydelliset sovitukset [25]

Puun täydellinen minimisovitus voidaan toteuttaa esimerkiksi Jack Edmondsin Blossom-algoritmilla [26, s. 1–2]. Sovituksen jälkeen paritetut solmut yhdistetään pienimpään virittävään puuhun, jolloin saadaan kuvan 13 mukainen verkko.

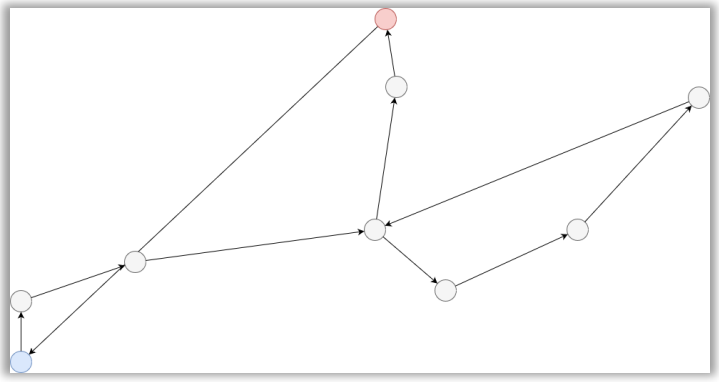


Kuva 13. Päivitetty verkko

Vaihe 3: Eulerin polun etsiminen

Täydellisen minimisovituksen jälkeen päivitetty verkko sisältää vain asteeltaan parillisia solmuja, joten se täyttää Eulerin polun määritelmän. Eulerin polku voidaan etsiä esimerkiksi Fleuryn algoritmilla. [15.]

Kuvassa 14 on päivitetystä verkosta löydetty Eulerin polku.



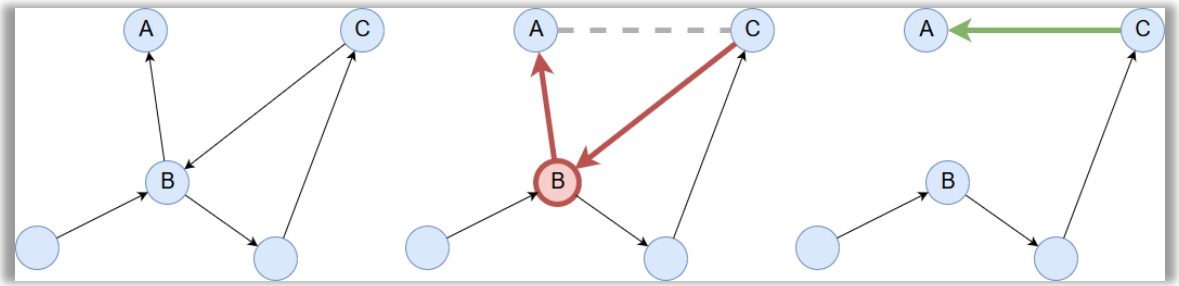
Kuva 14. Kuva Eulerin polusta

Vaihe 4: Hamiltonin polun luominen

Christofides-algoritmin viimeisenä vaiheena on Hamiltonin polun (engl. *Hamiltonian Path*) luominen. Jos Eulerin polussa kierretään kertaalleen kaikki verkon kaaret, niin Hamiltonin polussa kierretään kertaalleen kaikki solmut [27].

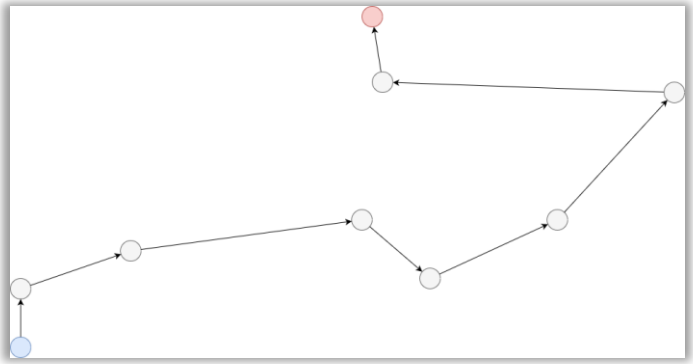
Hamiltonin polussa edetään Eulerin polun mukaan, mutta aiemmin vierailtuja solmuja ei oteta enää mukaan Hamiltonin polkuun. Tämä on sallittua, koska alkusyötteenä toiminut verkko täyttää kolmioepäyhtälön (engl. *triangle inequality*) määritelmän [20, s. 1].

Kolmioepäyhtälön määritelmän mukaan kolmion minkä tahansa sivun pituus on pienempi kuin jäljelle jäävän kahden sivun yhteenlaskettu pituus [28]. Tämä määritelmä on algoritmin tässä vaiheessa hyödyksi. Kuvassa 15 on havainnollistettu, kuinka solmujen A, B ja C muodostaman kolmion kaksi sivua poistamalla voidaan lyhentää reittiä. Poistaminen tapahtuu jättämällä solmu B toistamiseen vierailematta.



Kuva 15. Reitin lyhentäminen kolmioepäyhtälön avulla

Tässä kappaleessa esitetyn Christofides-algoritmin taustalla oli verkko, johon oli määritelty aloitus- ja lopetuspisteet. Jos niitä ei olisi ollut, olisi lopputuloksena ollut Hamiltonin kierros eli kaikki solmut kiertävä reitti, joka alkaa satunnaisesta pisteestä ja päätyy samaan pisteeseen. Kuvassa 16 on kuva Hamiltonin polusta ja valmiista reitistä.

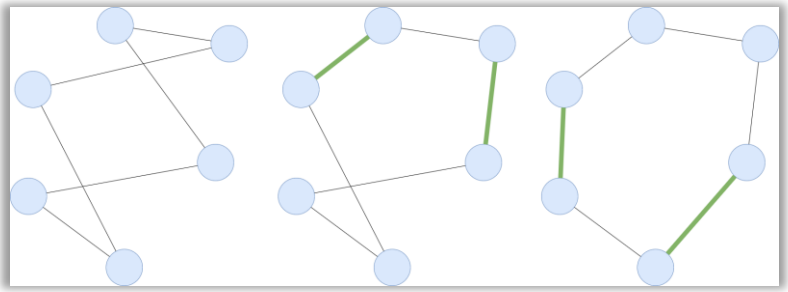


Kuva 16. Kuva Hamiltonin polusta ja valmiista reitistä

3.3.2 2-opt-algoritmi

2-opt-algoritmilla etsitään 2-optimaalista polkua verkon solmujen läpi. Polku on 2-optimaalinen, jos sille ei löydy lyhyempää 2-rinnakkaista polkua. Kaksi eri polkua määritellään 2-rinnakkaisiksi, jos toinen poluista pysytään muodostamaan toisesta kahta kaarta muokkaamalla. [29, s. 21–22.]

2-opt-algoritmin toiminta perustuu kaarien muokkaamiseen. Jokaisella kierroksella muokataan kahta kaarta kuvan 17 mukaisella tavalla. Jos muokkaus lyhentää muokattavaa polkua, niin se valitaan uudeksi poluksi ja aloitetaan uusi kierros. Jos muokkaus ei lyhennä polkua, niin kokeillaan muokata kaikkia kaaria, ja jos mikään muokkauksista ei ole muokattavaa polkua lyhyempi, niin se merkitään 2-optimaaliseksi ja keskeytetään algoritmi. [29, s. 22.]

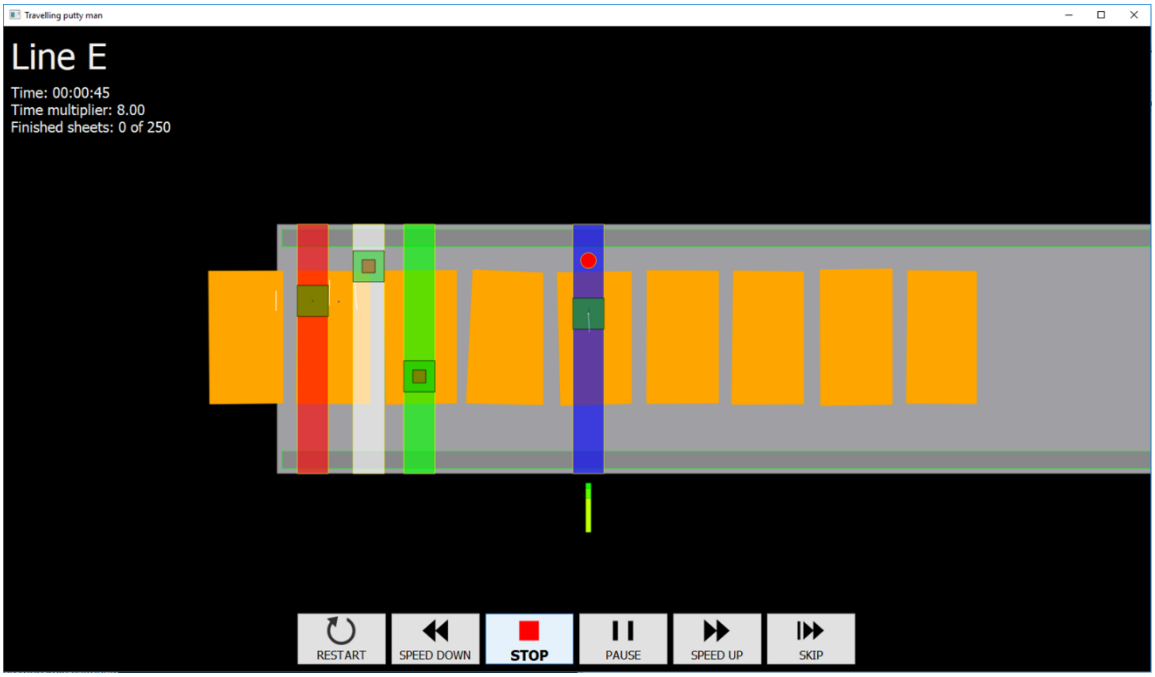


Kuva 17. 2-opt-algoritmin toiminta eri vaiheissa

4 Tuotantolinjan simulaattorin esittely

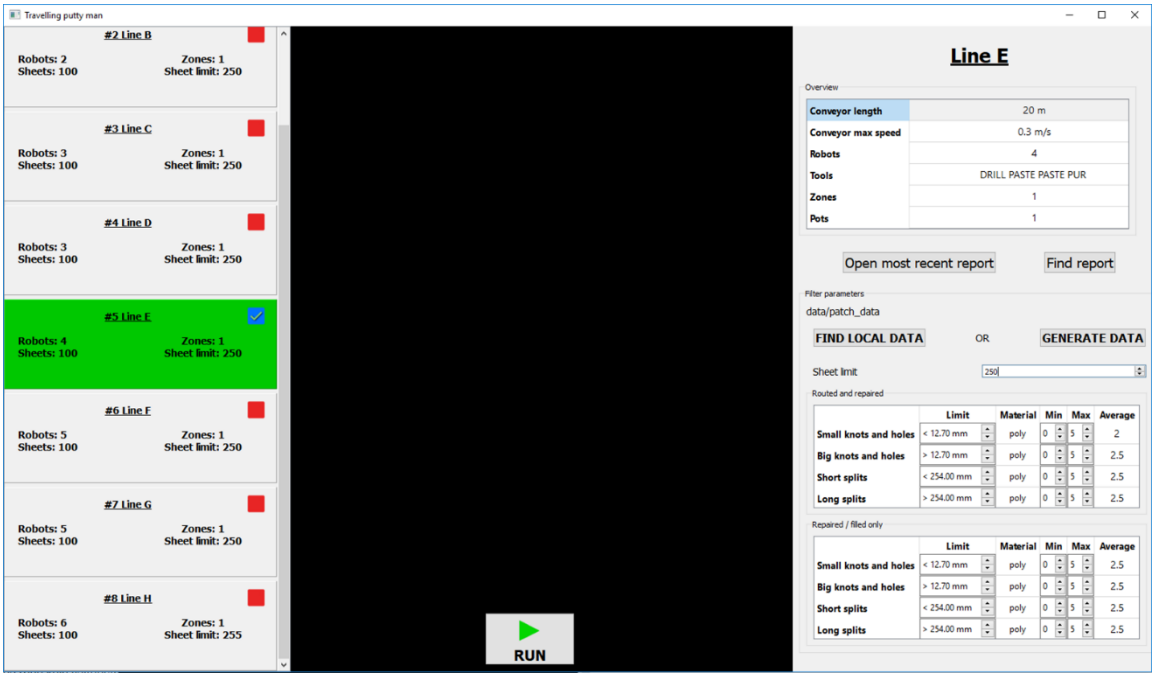
Isona osana tätä työtä on simulaattori, jolla voidaan simuloida työn kohteena olevan tuotantolinjan toimintaa. Simulaattori ja sen avulla kehitetyt optimointimenetelmät toimivat lähtölaukauksena tälle työlle, joten se esitellään lyhyesti tässä luvussa.

Kuvassa 18 on ruutukaappaus simulaattorista ajossa. Ruutukaappauksen keskellä oleva harmaa alue kuvaa linjaa, oranssit alueet tuotekappaleita ja punainen, valkoinen, vihreä ja sininen nelikulmio kuvaavat linjalla työskenteleviä robotteja. Ruutukaappauksen alaosassa on nähtävissä simulaattorin hallintapaneeli.



Kuva 18. Simulaattori ajossa

Kuvassa 19 on ruutukaappaus simulaattorin asetuksista. Ruutukaappauksen vasemmassa laidassa sijaitsee lista suoritettavista simulaatioista. Vihreällä merkitty simulaatio on aktiivisena ruutukaappauksen oikeassa laidassa, jossa on esitetty yhteenveto simulaation tiedoista ja asetuksista. Ruutukaappauksen alalaidassa on käynnistyspainike, joka suorittaa valitut simulaatiot.

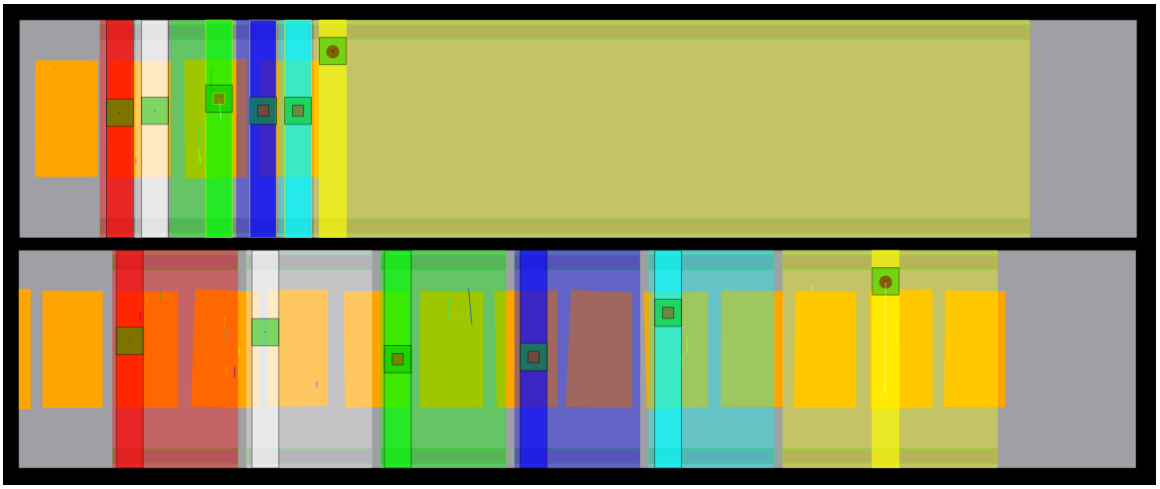


Kuva 19. Simulaattorin asetukset

4.1 Ominaisuudet

Simulaattorilla voidaan ajaa simulaatioita useilla erilaisilla robotti- ja työkalukombinaatioilla. Käyttäjä pystyy vapaasti lisäämään ja poistamaan robotteja ja muokkaamaan niiden ominaisuuksia. Jokaista simulaation kannalta merkittävää robotin ja työkalun parametria voidaan säätä erikseen.

Perinteisellä tuotantolinjalla on jokaisella robotilla omat, staattiset työskentelyalueet. Simulaattorissa käyttäjä voi kuitenkin halutessaan määritellä robotin työskentelyrajat liukuviksi. Liukuvilla työskentelyrajoilla varustetut robotit toimivat yhteisellä alueella, ottaen ja vapauttaen tilaa seuraavalle tilanteen mukaan. Kuvassa 20 on esimerkki liukuvista ja staattisista työalueista. Siinä osittain läpinäkyvillä väreillä värjätyt alueet kuvaavat samanvärisen robotin työaluetta. Kuvasta on helposti havaittavissa, miten liukuvilla työalueilla varustettu tuotantolinja käyttää koko linjan pituutta tehokkaammin hyödyksi.



Kuva 20. Esimerkki liukuvista (ylhäällä) ja staattisista (alhaalla) työalueista

Simulaattori käyttää korjauksiin aitoa tehtävädataa. Simulaatiossa ajettavat tuotantokappaleet on tallennettu joltain oikealta tuotantolinjalta. Käyttäjä voi halutessaan muokata tehtävädataa eri parametreilla. Käyttäjä voi esimerkiksi määritellä, että tuotantokappaleissa ei esiinny yhtään halkeamatehtävää.

Käyttäjä voi halutessaan nähdä visuaalisen näkymän simulaation kulusta. Visuaalisessa näkymässä voidaan reaaliajassa tarkastella, miten robotit suorittavat niille annetut tehtävät. Visuaalisen palautteen perusteella on helppo huomata mahdolliset logiikkavirheet linjan toiminnassa.

Simulaattorilla on mahdollista ajaa useaa simulaatiota rinnakkain. Jos käytössä on visualisointi, niin eri simulaatioiden välillä voi näkymää vaihtaa lennosta. Jokaisella simulaatiolla on omat ajonaikaiset parametrit, joten eri työkalukombinaatioita voidaan ajaa yhtä aikaa. Simulaattorissa on yksinkertainen työkalu eri ajojen vertailuun. Ajoja voidaan vertailla esimerkiksi suoritukseen kuluneen ajan, kapasiteetin, linjan hyötysuhteen ja robottien keskimääräisen työskentelyajan avulla.

Jokaisesta ajosta luodaan lisäksi raportti PDF-tiedostoon. Raportit sisältävät yksityiskohtaiset tiedot tuloksista, tilastotietoa käytetystä datasta ja kuvauksen linjan rakenteesta. Tuloksissa esitetään yhteenveto ajosta: suoritukseen kulunut aika, hyötysuhde ja tuotantokappaleiden, neliöiden ja kuutioiden määrät minuutissa. Käytetystä datasta esitetään tuotantokappaleiden määrät, niiden keskimääräiset koot, pinta-alat ja tilavuudet, tehtävien määrät ja tyypit sekä virheiden määrät. Linjan rakenteessa esitetään liukuhihnan ja robottien tilastot. Liukuhihnan tilastoissa on huomioitu liukuhihnan maksimi-, minimi- ja keski-

nopeudet sekä ajat pysähdyksissä ja täydessä vauhdissa. Robottien tilastoissa on huomioitu robottikohtainen työskentelyaika, suoritetut tehtävät, mahdollisesti käytetyn paikkausmateriaalin määrä sekä työkalukohtainen työskentelyaika.

4.2 Optimoinnin kehitystyökaluna

Yksi simulaattorin suurimpia vahvuuksia on toimia yhtenä kehitystyökaluna tämän työn toteutusvaiheessa. Simulaattori osaa antaa optimoinnista palautetta sekä visuaalisesti että numeerisesti: lasketun reitin järkevyyden (tai järjettömyyden) huomaa ihmissilmällä suhteellisen helposti ja tehtävien suorittamiseen kuluva aika on suora palaute optimoinnin onnistumisesta tai epäonnistumisesta. Tämän takia onkin olennaista, että simulaattori tukee suoraan tässä työssä kehitettävää optimointikirjastoa. Sen lisäksi käyttäjä voi määritellä jokaiselle ajolle omat parametrit, joiden perusteella optimointikirjasto suorittaa sen ajon tehtävien optimoinnin.

Olennaisena osana on myös tulosten vertailu. Koska ajoja voidaan simuloida rinnakkain eri parametreilla, voidaan raporttien avulla helposti vertailla, mikä optimoinneista on ollut tehokkainta. Tätä ominaisuutta käytetään tässä työssä tulosten vertailuun.

5 Optimointikirjaston toteutus

Optimointikirjaston toteutus on perinteinen ohjelmistonkehitysprosessi. Se jaetaan neljään vaiheeseen: vaatimusmäärittelyyn, suunnitteluun, toteutukseen ja testaukseen. Nämä vaiheet käsitellään yksityiskohtaisesti seuraavissa aliluvuissa.

5.1 Vaatimusmäärittely

Kehitettävä optimointikirjasto tulee osaksi olemassa olevaa ohjelmistoa, mikä asettaa luonnostaan hyvin yksityiskohtaiset vaatimukset optimointikirjaston toteutukselle. Kirjaston käyttöliittymän toteutuksessa käytetään ennestään valmiita komponentteja datan siirtämiseen. Ohjelmointiympäristönä käytetään QtCreatoria ja ohjelmointikielenä C++:aa. Tulevaisuutta silmällä pitäen optimointikirjaston kehityksessä ei saa käyttää Qt:n omia kirjastoja, vaan ainoastaan C++:n standardikirjastoa ja kohdeyrityksen omia kirjastoja. Kehitysalustana toimii ensisijaisesti Linux, mutta optimointikirjaston on toimittava myös Windows-alustoilla.

Tärkeä vaatimus on myös optimointiin käytettävissä olevan ajan huomioiminen. Tuotantolinja on koko ajan liikkeessä ja optimoitavia tehtäviä tulee jatkuvalla syötteellä, joten yhden tuotantokappaleen tehtävien optimoimiseen ei ole käytettävissä paljon aikaa. Käytettävissä oleva aika riippuu kuitenkin tuotantolinjan nopeudesta: nopeammin kulkevalla linjalla optimointiin käytettävä aika on luonnollisesti pienempi. Koska tätä ei voida etukäteen määritellä, on optimointikirjastoon luotava mahdollisuus optimointiin käytettävän maksimiajan määrittämiseen.

Optimointikirjaston koodin optimoiminen ei kuitenkaan ole tämän työn vaatimuksena, koska kirjaston suoritusaikoja ei voida testata oikealla linjalla. Se jätetäänkin tulevaisuuteen. Työn vaatimuksena on enemmänkin sellaisen optimointikirjaston luonti, joka tuottaa hyviä ratkaisuja tehtävien optimointiin riippumatta käytetystä ajasta.

Optimoinnin suorituksessa on vaatimuksena vanhan optimoinnin mukainen menetelmä, jossa yhden tuotantokappaleen kaikki tehtävät lähetetään eteenpäin yhdellä kertaa. Simulaattorin kehityksen aikana kuitenkin luotiin konsepti mallista, jossa tehtävät lähetettäisiin yksi kerrallaan. Tämä huomioitiin vaatimuksessa siten, että konsepti pidetään mielessä optimoinnin rakennetta suunniteltaessa, vaikka sitä ei olekaan tarkoitus välittömästi toteuttaa. Konsepti on selitetty yksityiskohtaisemmin kehitysideoissa luvussa 6.4.

5.2 Suunnittelu

Vaatimusmäärittelyn perusteella aloitetaan kirjaston suunnittelu. Koska tämä optimointikirjasto tulee osaksi olemassa olevaa ohjelmistoa, on sen käyttöliittymä suunniteltava toimimaan yhdessä muun ohjelmiston kanssa. Käyttöliittymään luodaan erilliset funktiot optimointikirjaston alustamiseen, tehtävien optimointiin ja optimointikirjaston sulkemiseen.

Tehtävien optimoinnissa huomioidaan erityisesti jo mainitut kaksi isoa optimointiongelmaa: tehtävien jakaminen roboteille ja työjärjestyksen optimointi. Näihin pitää olla idea mahdollisista ratkaisuista ennen kehittämisen aloittamista. Tässä työssä esiteltävien ratkaisujen pohjana toimii kohdeyrityksen vanha optimointikirjasto ja simulaattorin kehityksen yhteydessä tehdyt havainnot.

5.3 Kehitys

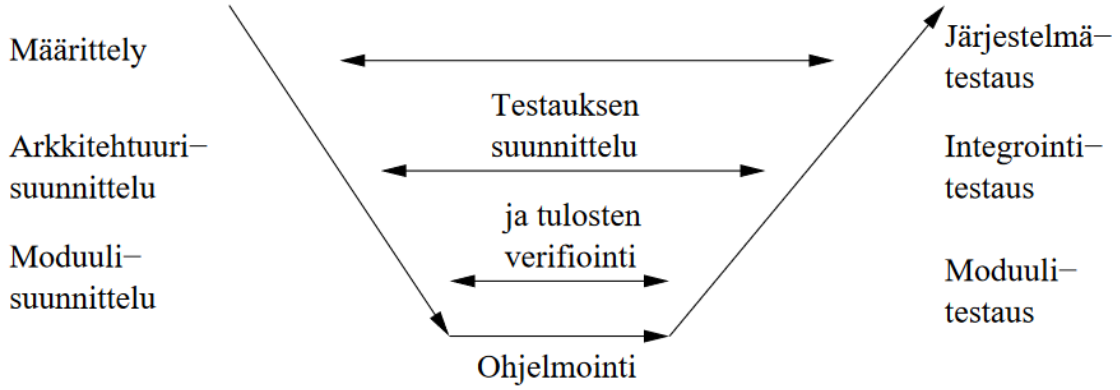
Suunnittelun jälkeen aloitetaan optimointikirjaston kehitys. Kehityksen ensimmäisenä tavoitteena on luoda käyttöliittymän rakenne ja saada optimointikirjasto toimimaan testi- ja kehitysympäristön kanssa. Kun toiminta on varmistettu, voidaan lähteä kehittämään kirjaston ominaisuuksia.

Optimointikirjaston sisäisen rakenteen lähtökohtana oli lähdekoodin helppo muokattavuus ja laajennettavuus. Rakennetta ei kuitenkaan erityisemmin suunnitella etukäteen, vaan työssä käytetään hyväksi Casey Muratorin oppeja kompressoivasta ohjelmistonkehityksestä: tehdään ensin yksinkertaisin toimiva toteutus yhteen lähdekooditiedostoon, ja kun ohjelmiston rakenne ja toimivuus on varmistunut, niin lähdekoodi voidaan tarvittaessa luontevasti pilkkoa helpommin käytettäviin palasiin [30]. Näin ohjelmisto rakentuu loogisesti pala kerrallaan ja jos toimivuudessa huomataan puutteita, niin ne pystytään paikallisesti korjaamaan.

5.4 Testaus

Optimointikirjasto tullaan testaamaan kuvassa 21 esitetyn V-mallin mukaisesti. Siinä testaus on jaettu kolmeen osaan: järjestelmätestaukseen, integrointitestaukseen ja moduuli-

testaukseen. Järjestelmätestauksessa testataan koko järjestelmä ja sen suorituskyky, integrointitestauksessa moduulien yhteistoiminta ja moduulitestauksessa yksittäisen moduulin toiminta. [31, s. 35.]

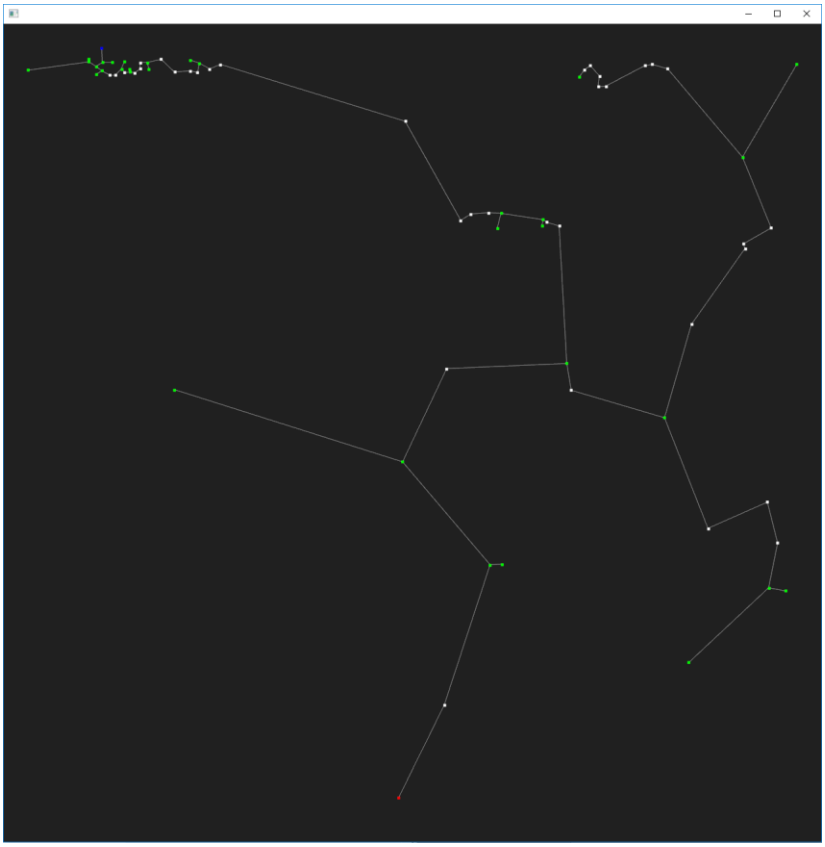


Kuva 21. Testauksen V-malli [32, s. 9]

5.4.1 Moduulitestaus

Moduulien testaukseen voidaan käyttää sekä sisäistä testausta (engl. *structural testing* tai *white-box testing*) että ulkoista testausta (engl. *functional testing* tai *black-box testing*). Sisäisessä testauksessa käydään läpi ohjelman lähdekoodia ja testataan sen oikeellisuus, kun taas ulkoisessa testauksessa tarkastellaan moduulin tuloksien oikeellisuus syöttöarvoihin verrattuna. [31, s. 36.]

Tämän työn aiheena olevaa optimointikirjastoa voidaan pitää yhtenä moduulina, joka keskustelee ohjelmiston muiden moduulien kanssa. Optimointikirjaston sisäistä testausta tehdään jatkuvasti kehityksen yhteydessä testaamalla ohjelmakoodin logiikkaa kehitysympäristön debug-työkaluilla. Ulkoinen testaus suoritetaan kehitysvaiheessa numeerisesti ja silmämääräisesti: moduulin tuottamia tuloksia verrataan lähtöarvoihin ja tarkastellaan visuaalisen palautteen avulla. Erityisesti Christofides-algoritmin kehitystä varten luotiin erillinen testaussovellus, josta on esimerkki kuvassa 22.



Kuva 22. Esimerkki Christofides-algoritmin testaussovelluksesta

5.4.2 Integrointitestaus

Tässä työssä integrointitestausta suoritetaan erityisesti käyttöliittymän suunnittelun ja kehityksen yhteydessä. Siinä vaiheessa testataan, että tarvittava data kulkee moduulien välillä ja oikeassa muodossa. Testausta suoritetaan myös säännöllisesti kehityksen yhteydessä. Moduulien yhteistoiminta todetaan sekä simulaattorilla että muulla ohjelmistolla. Testidatana käytetään dataa oikealta tuotantolinjalta.

5.4.3 Järjestelmätestaus

Tämän työn järjestelmätestaus voidaan suorittaa kahdella tavalla: joko virtuaalisesti simulaattorilla tai oikealla tuotantolinjalla. Ensisijaisena testausmenetelmänä toimii simulaattori, sillä se on aina saatavilla ja sen antamat tulokset ovat vertailukelpoisia keskenään. Tämä on kehitysvaiheessa riittävä testausmenetelmä. Lopullinen testaus suoritetaan oi-

kealla tuotantolinjalla, jossa ovat mukana kaikki linjan komponentit. Silloin voidaan oikeasti nähdä ja todeta optimoinnin suorituskyky. Testaus oikealla tuotantolinjalla on kuitenkin hankalaa, sillä niille on kehitysvaiheessa hyvin rajattu pääsy. Testaus oikealla tuotantolinjalla jätetäänkin tulevaisuuteen.

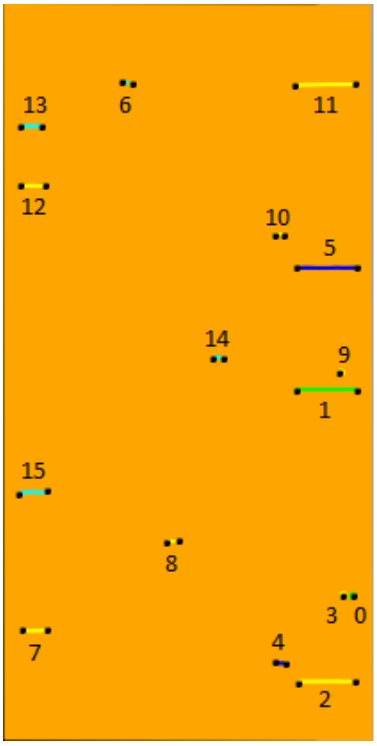
5.4.4 Lopputestaus

Toteutusvaiheen lopulla suoritetaan simulaattorilla erillinen lopputestaus, jossa vertaillaan kehitetyn optimoinnin suorituskykyä vanhaan optimointiin. Lopputestaus ja sen tulokset esitetään luvussa 6.

5.5 Optimoinnin vaiheet

Itse optimoinnin toimintalogiikka voidaan jakaa useaan eri vaiheeseen: alustukseen, tehtävien lukemiseen, tehtävien jakamisen optimointiin, työjärjestyksen optimointiin ja lopetukseen. Alustus- ja lopetusvaiheet suoritetaan muun ohjelmiston alustuksen ja lopetuksen yhteydessä. Tehtävien lukeminen, tehtävien jakamisen optimointi, työjärjestyksen optimointi ja tehtävälistan luonti suoritetaan silloin, kun uusia tuotteita ja tehtäviä saapuu linjalle.

Seuraavissa aliluvuissa esitetään optimoinnin kulku tarkemmin esimerkkien avulla, kun lähtökohtana on kuvan 23 mukainen tilanne. Kuvassa on nelikulmion muotoinen tuotantokappale (oranssi) jolla on muutamia tehtäviä (mustat pisteet ja niitä yhdistävät värilliset viivat). Viivoilla yhdistetyt tehtävät ovat ns. halkeamia. Kuvassa esiintyvä data pohjautuu oikealla linjalla mitattuihin arvoihin.



Indeksi	X	Y	Tyyppi	Työkalu	Tilavuus
0	2170	62	1	4	17
1	1494	148	3	4	4278
2	2445	155	3	3	5100
3	2167	94	1	3	26
4	2389	301	2	4	3765
5	1097	149	3	4	4496
6	495	807	2	4	3731
7	2282	1106	3	3	2083
8	1990	653	2	3	3752
9	1442	102	1	3	15
10	990	305	2	3	3788
11	501	151	3	3	4443
12	833	1108	3	3	1879
13	638	1110	2	4	2016
14	1391	503	2	4	3616
15	1831	1107	3	4	2052

Kuva 23. Tuotantokappale ja siihen liittyvät tehtävät

5.5.1 Alustus

Optimointikirjasto tarvitsee tiedot linjalla toimivista roboteista ja niiden käytössä olevista työkaluista. Nämä voidaan lukea joko suoraan linjalta tai simulaattorin tapauksessa syöttää käsin.

Alustusvaiheessa ladataan optimoinnissa käytettävät parametrit. Parametrit ovat arvoja, joita muokkaamalla optimointia voidaan hienosäätää käyttäjän haluamaan suuntaan. Käyttäjä voi esimerkiksi määritellä, millä perusteella tehtävät jaetaan roboteille ja miten reitinhaku toteutetaan.

Alustuksen yhteydessä nollataan optimointia tukevat ajonaikaiset tilapäistiedot ja historiatiedot. Jos alustus onnistuu, on optimointikirjasto suoritusvalmis. Muuten alustuksen epäonnistumisesta lähetään virheilmoitus käyttäjälle eikä optimointia voida suorittaa.

5.5.2 Tehtävien lukeminen

Optimointikirjasto voi vastaanottaa tehtäviä heti kun se on hyväksytysti alustettu. Tehtävät tulevat kohdeyrityksen muilta ohjelmistoilta objektidatana. Objektidata on kohdeyrityksen sisäinen tiedostomuoto, joka sisältää runsaasti tietoa optimoinnin kohteena olevasta tuotteesta ja sille suoritettavista tehtävistä. Erityisesti tehtävien suorittamiseen vaadittavaa tietoa käytetään optimoinnissa.

Tehtäväkohtaiset tiedot on tallennettu objektidataan tietueina. Näitä tietueita voitaisiin sellaisenaan käyttää tulevissa optimointivaiheissa, mutta ne sisältävät runsaasti optimoinnin kannalta merkityksetöntä tietoa. Välimuistin yhtenäisyyden (engl. *cache coherence*) takia on perusteltua luoda uusi kevyempi tietue, josta löytyisivät vain optimoinnin kannalta oleelliset tiedot. Tehtävien tietoja luettaessa ja uutta tietuetta luotaessa tärkeitä arvoja ovat tehtävän keskipisteen sijainti, tehtävän työpisteet, tehtävän tyyppi, käytettävän työkalun tyyppi ja tehtävän mahdollinen tilavuus.

Optimoinnin näkökulmasta tehtävät ovat kaksiulotteisia ja sijaitsevat kaksiulotteisessa koordinaatistossa. Tehtävän keskipisteen sijainti on jokin piste tuotantokappaleen pinnalla, ja tehtävän työpisteet ovat tehtävän suorittamiseen vaadittavat koordinaatit: tehtävää suorittavan robotin on käytävä kaikissa näissä pisteissä. Jokaista työpistettä kohden on mahdollinen tilavuusarvo. Jos se on asetettu, niin tehtävää suorittavan robotin on esimerkiksi käytettävä tehtävään kyseisen tilavuuden verran paikkausmateriaalia. Tehtävään käytettävä materiaali riippuu käytettävän työkalun tyypistä.

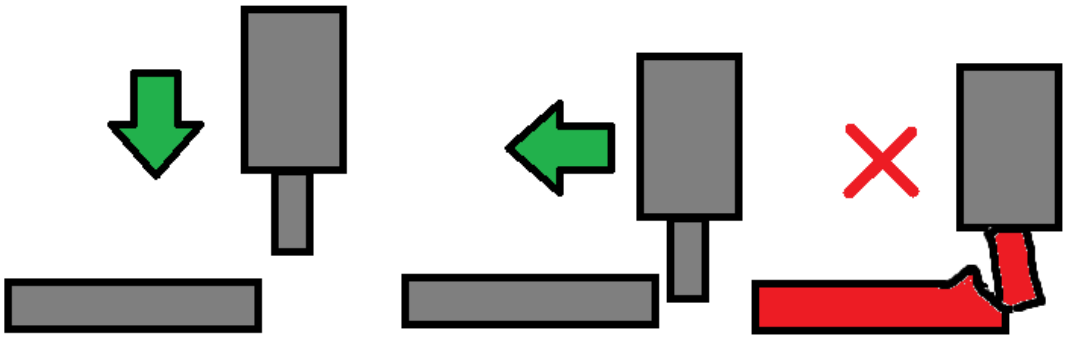
Tehtävässä käytettävän työkalun tyypin perusteella tehtävä voidaan seuraavassa vaiheessa jakaa roboteille. Jos tuotantolinjalta ei löydy kyseistä työkalua käyttävää robottia, niin tehtävää ei voida suorittaa eikä sitä oteta mukaan optimointiin. Tehtävän tyyppi kuvaa lähinnä tehtävän visuaalista ilmettä. Se voi olla yksi piste, halkeama (normaali tai yksinkertainen), ellipsi, nelikulmio, spiraali tai täyttöspiraali. Tehtävän tyypit on esitetty taulukossa 1.

Yksi piste. Tehtävä jolla on vain yksi työpiste.	
Yksinkertainen halkeama. Sijaitsee kahden työpisteen välillä.	
Halkeama. Kuin yksinkertainen halkeama, mutta työpisteitä voi olla useampi.	
Ellipsi. Edistynyt ellipsin muotoinen tehtävä, joka näkyy optimoinnissa yhtenä työpisteenä. Robotti suorittaa liikkeen itsenäisesti.	
Nelikulmio. Kuten ellipsi, mutta nelikulmion muotoinen. Robotti suorittaa liikkeen itsenäisesti.	
Spiraali. Nelikulmio, joka koostuu useista työpisteistä, jotka muodostavat spiraalimaisen reitin.	
Täyttöspiraali. Kuin spiraali, mutta ei nelikulmion muotoinen. Käytetään vain erikoistapauksissa.	

Taulukko 1. Tehtävien tyyppien selitykset

Optimoinnin kannalta tehtävän tyypillä on merkitystä ainoastaan halkeamien osalta. Halkeamien korjaussuunnalla on iso merkitys erityisesti optimaalista reittiä etsittäessä, mutta se pitää huomioida myös tehtävien lukuvaiheessa. Jos halkeamat sijaitsevat tuotantokappaleen reunoilla, saattaa olla mielekästä, että halkeaman työskentelysuunta on kohti reunaa. Jos työskentelysuunta on reunalta sisäänpäin, saatetaan joutua kuvan 24 mukaiseen

tilanteeseen. Siinä työkalu on laskettu hieman tuotantokappaleen ulkopuolelle, minkä seurauksena työkalu on liikkeen alkaessa törmännyt kappaleeseen vaurioittaen molempia. Käyttäjä voikin parametrilla määrittää reunaa lähellä olevien halkeamien työskentelysuunnan.



Kuva 24. Esimerkki lähellä reunaa olevien halkeamatehtävien haasteista

Tulevaisuudessa tehtävän tyypillä lienee isompi merkitys, kun tehtävän suoritusajan arviointi saadaan toteutettua. Sen laskemiseen on oleellista tietää tehtävän tyyppi, sillä esimerkiksi pistemäisen tehtävän ja spiraalimaisen tehtävän välinen suoritusaikaero on huomattava.

Reunalla sijaitsevien halkeamien kääntämisen jälkeen on aika aloittaa itse optimointiprosessi. Sen ensimmäinen vaihe on tehtävien jakamisen optimointi.

5.5.3 Tehtävien jakaminen

Tässä vaiheessa jaetaan luetut tehtävät tuotantolinjalla toimiville roboteille. Tehtävät on tarkoitus jakaa siten, että kaikki samalla työkalulla suoritettavat tehtävät on jaettu tasaisesti samaa työkalua käyttäville roboteille kuvan 23 (s. 28) mukaisesti.

Tehtävien jakaminen voidaan edelleen jakaa kolmeen vaiheeseen: tehtävien lajittelemiseen työkalun tyypin mukaan, tehtävien lajitteleminen työkalukohtaisesti ja tehtävien jakaminen työkalukohtaisesti. Kaikkiin lajittelutehtäviin käytetään C++:n standardikirjaston sort-lajittelufunktiota. Sen sisäinen toteutus vaihtelee, mutta yleisimmissä tilanteissa käytetään pikalajittelua. Tähän työhön sort-lajittelunfunktion nopeus on riittävä, mutta se voidaan tarvittaessa korvata nopeammalla lajittelualgoritmilla.

Tehtävien lajitteleminen työkalun tyypin mukaan

Tehtävät luettiin lukuvaiheessa yhteen taulukkoon, jossa ne ovat satunnaisessa järjestyksessä. Ensimmäisenä vaiheena on tehtävien lajittelu siten, että ne ovat järjestyksessä työkalun tyypin mukaan. Tämän vaiheen jälkeen samalla työkalulla suoritettavat tehtävät sijaitsevat muistissa peräkkäin, jolloin niitä voidaan käsitellä tehokkaasti yhdellä for-silmukalla. Taulukossa 2 on tehtävät jaettu työkalun tyypin mukaan.

Järjestys	X	Y	Tehtävä	Työkalu
2	237	244	2	1
3	295	32	1	1
5	219	169	1	1
7	178	85	1	1
9	147	143	1	1
10	117	22	2	1
12	71	170	1	1
14	41	60	1	1
1	335	100	1	2
4	231	91	2	2
6	219	262	1	2
8	154	214	1	2
11	90	95	1	2
13	55	278	2	2

Taulukko 2. Tehtävät työkalun tyypin mukaan lajiteltuna

Tehtävien lajitteleminen työkalukohtaisesti

Kun tehtävät on lajiteltu työkalutyypin mukaan, ne lajitellaan vielä edelleen käyttäjän määrittelemän parametrin mukaan. Käyttäjä voi valita, lajitellaanko tehtävät pituus- tai leveyssuunnassa vai tehtävien tilavuuden mukaan. Tämä lajittelu toimii tärkeänä esiasteena seuraavalle työvaiheelle. Taulukossa 3 on tehtävät jaettu työkalun tyypin ja pituussuunnan mukaan.

Järjestys	X	Y	Tehtävä	Työkalu
14	41	60	1	1
12	71	170	1	1
10	117	22	2	1
9	147	143	1	1
7	178	85	1	1
5	219	169	1	1
2	237	244	2	1
3	295	32	1	1
13	55	278	2	2
11	90	95	1	2
8	154	214	1	2
6	219	262	1	2
4	231	91	2	2
1	335	100	1	2

Taulukko 3. Tehtävät työkalun tyypin ja pituussuunnan mukaan lajiteltuna

Tehtävien jakaminen työkalukohtaisesti

Tässä vaiheessa tehtävät jaetaan työkalukohtaisesti roboteille. Jokaisella työkalutyypillä on tieto roboteista, joilta kyseinen työkalutyyppi löytyy. Käyttäjä valitsee parametrilla tehtävän avainarvon, joka määrittelee, miten suoritettavat tehtävät roboteille jaetaan: tasaisesti joko alueittain pituus- tai leveyssuunnassa, lukumäärän mukaan, tilavuuden mukaan tai työskentelyaika-arvion mukaan.

Ennen jakamista suoritetaan alkutoimenpiteet. Ensin valitaan robotti, jolle tehtävien jakaminen aloitetaan. Sitten lasketaan robottikohtainen työraja, jonka verran työtä yhdelle robotille voidaan jakaa. Työraja lasketaan tehtävien yhteenlasketun avainarvon mukaan jaettuna robottien määrällä.

Jakaminen suoritetaan for-silmukassa tehtävä kerrallaan. Jokaisen tehtävän kohdalla annetaan kyseinen tehtävä valitulle robotille ja tarkistetaan sen työkuorma. Valittu robotti vaihdetaan seuraavaan, jos työkuorma nousee yli robottikohtaisen työrajan.

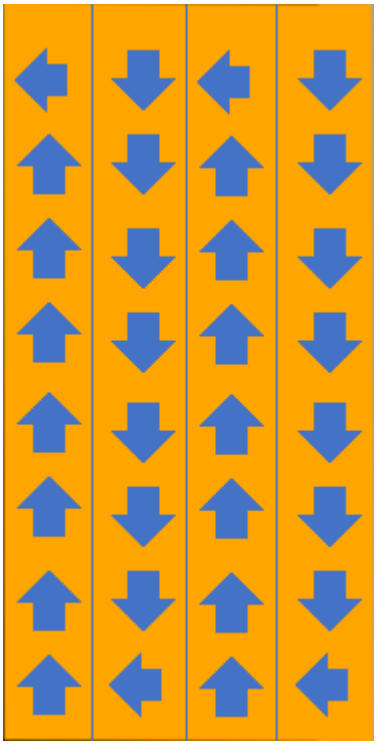
5.5.4 Työjärjestyksen optimointi

Optimoinnin seuraava haaste on robottikohtaisen työjärjestyksen optimointi. Edellisen vaiheen jäljiltä jokaisen robotin tehtävät ovat tiedossa. Tässä vaiheessa on tarkoitus lajitella

jokaisen robotin tehtävät siten, että niiden suoritukseen kuluu mahdollisimman pieni aika. Tämäkin vaihe voidaan jakaa useampaan pienempään osaan: tehtävien jakamiseen ryhmiin, reitin aloitus- ja lopetuspisteiden hakuun ja reitinhakuun.

Tehtävien jakaminen ryhmiin

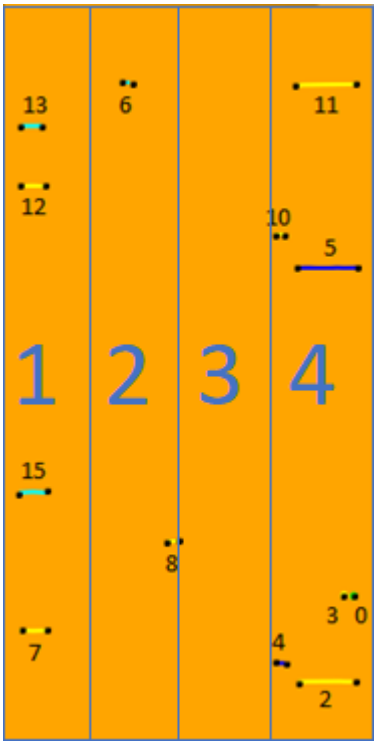
Yksi vanhasta optimoinnista säilytetty ominaisuus on robotin tehtävien jakaminen pienempiin ryhmiin. Ryhmien yksi tavoite on reitinhaun nopeuttaminen: jos reittiä haetaan kaikille tehtäville yhdellä kertaa, voi laskenta-aika olla huomattavan pitkä. Tehtävien jakaminen pienempiin ryhmiin tarkoittaa vähemmän reittivaihtoehtoja ja nopeampia laskenta-aikoja. Ryhmien toinen tavoite on reitin hallinta: käyttäjä voi esimerkiksi haluta, että reitti etenee matomaisesti tuotteen pinnalla kuvan 25 mukaisesti, mikä vapauttaa jatkuvasti liikkuvan tuotteen pinta-alaa seuraavalle robotille.



Kuva 25. Reitin eteneminen matomaisesti

Käyttäjä voi itse määritellä ryhmien minimi- ja maksimimäärän, mutta niitä on aina vähintään yksi. On myös hyvä huomioida, että vanhassa optimoinnissa ryhmien lukumäärä oli kaikille roboteille sama, mutta tässä se voi vaihdella; esimerkiksi robotilla A voi olla kaksi ryhmää ja robotilla B voi olla neljä ryhmää. Käyttäjä voi lisäksi määritellä ryhmissä sijaitsevien tehtävien tavoitemäärän.

Ryhmät luodaan toistaiseksi tuotantokappaleen pinta-alasta kuvan 26 mukaisesti. Kaikki ryhmät ovat nelikulmioita tuotantokappaleen pinnalla, ja ne sisältävät niiden reunojen sisäpuolelle jäävät tehtävät.



Kuva 26. Tehtävien jako neljään ryhmään

Kun tehtävät on jaettu ryhmiin, niin suoritetaan ryhmien sisäinen reitinhaku. Sen esivaiheena toimii reitin aloitus- ja lopetuspisteiden haku.

Reitin aloitus- ja lopetuspisteiden haku

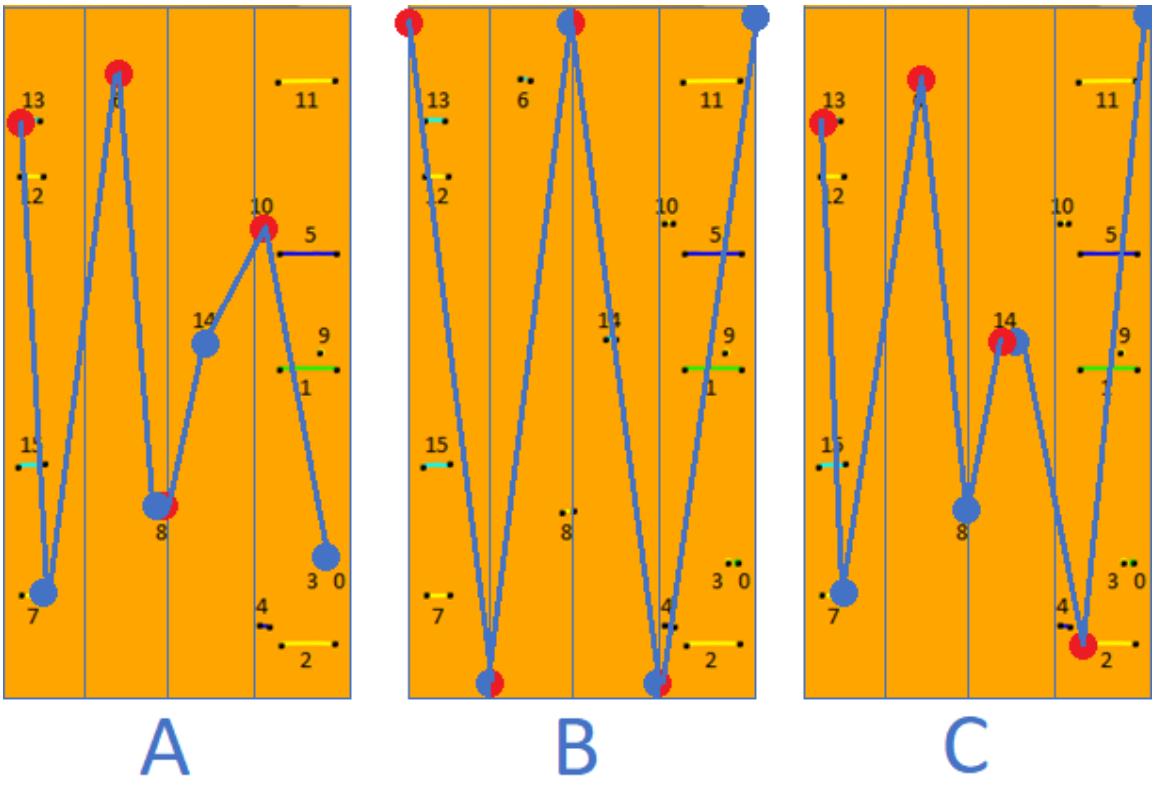
Ennen reitinhakua haetaan aloitus- ja lopetuspisteet reitille. Aloitus- ja lopetuspisteillä on tavoitteena saada robotin kulkemaan reittiin johdonmukaisuutta ja parantaa reitinhaun lopputulosta. Tarkoituksena on, että reitti alkaisi läheltä aloituspistettä ja päättyisi lähelle lopetuspistettä.

Aloitus- ja lopetuspisteet voidaan etsiä usealla eri algoritmilla, joista käyttäjä voi valita yhden. Näitä ovat aloitus- ja lopetuspisteiden haku kulkusuunnan mukaan, lähimmän kulman mukaan ja lyhyimmän etäisyyden mukaan.

Aloitus- ja lopetuspisteiden haku kulkusuunnan mukaan on näistä yksinkertaisin. Aloituspisteeksi valitaan yksinkertaisesti ryhmän kulkusuunnan puoleinen reuna ja lopetuspisteeksi ryhmän tulosuunnan puoleinen reuna. Tavoitteena on luoda seuraavassa vaiheessa reitti, joka alkaa mahdollisimman läheltä kulkusuunnan puoleista reunaa ja päättyy mahdollisimman lähelle tulosuunnan puoleista reunaa kuvan 27 A-kohdan mukaisesti.

Aloitus- ja lopetuspisteiden haku lähimmän kulman mukaan on myös perua vanhan optimoinnin matomaisesta liikkeestä. Aloituspisteeksi valitaan aina edellisen ryhmän lopetuspistettä lähinnä oleva kulma ja lopetuspisteeksi aloituspistettä vastakkainen kulma. Tavoitteena on reitti, joka kulkisi matomaisesti tuotteen pinnalla kuvan 27 B-kohdan mukaisesti.

Aloitus- ja lopetuspisteiden haku lyhyimmän etäisyyden mukaan on kehitteillä oleva algoritmi. Siinä on tarkoituksena ottaa aloituspisteeksi suoraan edellisen ryhmän lopetuspiste, mikä takaa sujuvan liikkeen ryhmien välillä. Sen haasteena on löytää hyvä logiikka ryhmän lopetuspisteen määrittelemiseksi. Tällä hetkellä lopetuspisteeksi valitaan aloituspisteestä kauimpana oleva piste kuvan 27 C-kohdan mukaisesti, mikä ei ole optimaalinen ratkaisu.



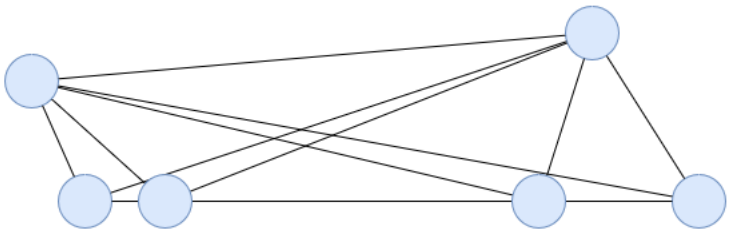
Kuva 27. Esimerkki eri aloitus- ja lopetuspisteiden valinnoista

Kun aloitus- ja lopetuspisteet ovat selvillä, suoritetaan reitinhaku.

Reitinhaku

Reitinhaku on optimointivaiheen eniten aikaa vievä vaihe. Tehtäviä, joiden välille reitti lasketaan, voi olla useita kymmeniä, mahdollisesti jopa yli sata. Tehtävien suuren määrän vuoksi raa'an voiman käyttö laskentaan on useissa tapauksissa poissuljettu. Käyttäjä voi erikseen määrittää, millä tehtävämäärällä raaka voimaa voidaan vielä käyttää, mutta muuten reitit lasketaan Christofides-algoritmilla. Christofides-algoritmi on tarkemmin kuvattu kappaleessa 3.3.1.

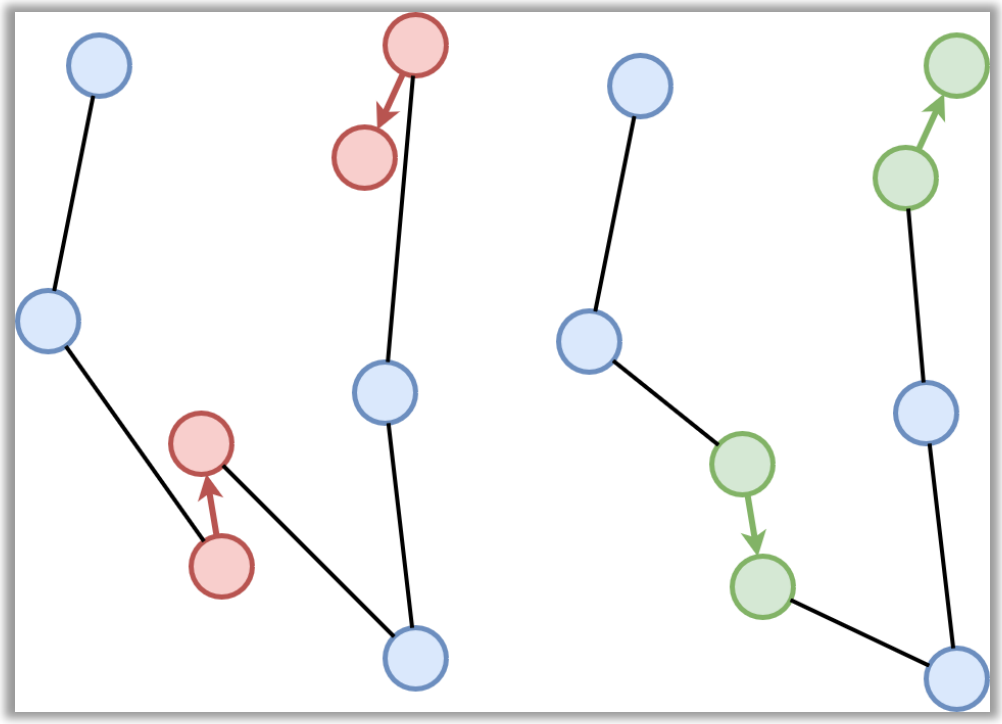
Christofides-algoritmin alkusyötteenä toimii verkko, jossa on reitin jokaisen työpisteen etäisyys muihin työpisteisiin kuvan 28 mukaisesti. Verkosta haetaan edelleen Primin algoritmilla pienin virittävä puu ja paritonasteisille solmuille etsitään parit. Aloitus- ja lopetuspisteet yhdistetään toisiinsa, jolloin saadaan tulokseksi Eulerin polku. Hamiltonin algoritmilla poistetaan reitistä useammin toistuvat pisteet ja saadaan lopullinen reitti. Christodies-algoritmin lisäksi käytetään 2-opt-algoritmia reitin parantamiseen.



	1	2	3	4	5	6
1	0	1502	502	439	1227	1427
2	1502	0	1805	1361	426	335
3	502	1805	0	451	1449	1645
4	439	1361	451	0	998	1194
5	1227	426	1449	998	0	200
6	1427	335	1645	1194	200	0

Kuva 28. Työpisteiden muodostama verkko ja etäisyysmatriisi

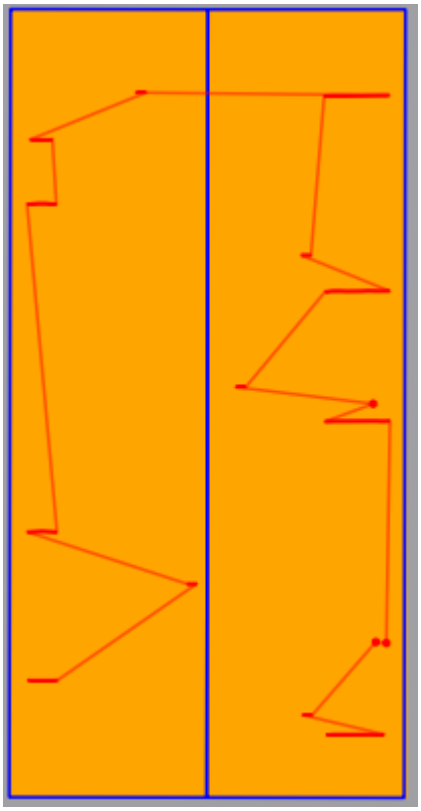
Ennen lopullista reittiä suoritetaan halkeamien kulkusuunnan kääntöoperaatio kuvan 29 mukaisesti. Kuvassa punaiset ja vihreät solmut ja niitä yhdistävät nuolet esittävät halkeamia. Jos halkeaman kulkusuunta on todellista reitin kulkusuuntaa vastaan, niin se on järkevää kääntää toiseen suuntaan. Tätä ei tietenkään voida soveltaa lähellä reunaa sijaitseviin halkeamiin, sillä ne on käännetty jo tehtävien lukemisen yhteydessä.



Kuva 29. Halkeamien kulkusuunnan kääntäminen

Halkeamat ovat reitinhaun kannalta muutenkin ongelmallisia. Jotta reitti olisi optimaalinen, pitää sitä laskettaessa jo tietää, kumpaan suuntaan halkeamaa työstetään. On myös olemassa rinnakkaisia tehtäviä, jotka koostuvat useista halkeamista. Niiden kääntäminen ja optimointi jätetään jatkokehitykseen.

Reitinhaun lopputuloksena on kuvan 30 mukainen reitti.



Kuva 30. Lopullinen reitti

5.5.5 Tehtävälistan luonti

Optimoinnin viimeinen työvaihe on tehtävälistan luonti. Kaikki optimoinnissa mukana olleet tehtävät lisätään yhteen listaan ja palautetaan tuotantolinjan ohjelmistolle, joka välittää tehtävät lopulta roboteille.

5.5.6 Lopetus

Optimointikirjaston lopetusvaiheen ainoa toiminnallisuus muistin siivoamisen lisäksi on kehitysvaiheessa käytetyn tiedon tallentaminen. Optimointikirjasto osaa mitata funktioiden suorittamiseen kuluvat ajat, laskea niiden keskiarvot ja tallentaa ne tiedostoon tarkempaa tarkastelua varten. Optimointikirjasto pitää myös kirjaa mahdollisista virheilmoituksista.

6 Tulokset

Optimointikirjaston toteutuksen arvioimiseksi suoritetaan lopuksi vielä lopputestaus simulaattorilla. Testauksessa on tarkoituksena verrata uutta ja vanhaa optimointia. Vertailuperusteena käytetään simulaattorilta luettavaa tuotantolinjan hyötysuhdetta, joka kuvastaa linjan toteutunutta kapasiteettia suhteessa maksimikapasiteettiin.

Toteutuneella kapasiteetilla tarkoitetaan linjan läpi kulkevien tuotantokappeleiden määrää minuutissa, ja linjan maksimikapasiteetti kuvaa kapasiteettia optimitilanteessa, jossa linja etenee maksimivauhdilla ilman hidastuksia. Simulaattorilla kapasiteetin laskeminen aloitetaan, kun ensimmäinen tuotantokappale on poistunut linjalta, ja lopetetaan, kun viimeinen tuotantokappale on saapunut linjalle. Näin kapasiteettilaskuissa ei huomioida tuotannon alun ja lopun tilanteita, joissa linja on pitkältä matkalta tyhjä.

6.1 Testauksen määrittely

Testilinjana käytetään yhtä kuuden robotin linjaa, joka pohjautuu oikeaan tuotantolinjaan. Testidatana käytetään samalta tuotantolinjalta tallennettua dataa. Testidataa on kolme pakettia: A, B ja C. Jokaisessa paketissa on tehtäviä kaikille roboteille, mutta tuotantokappaleiden määrä vaihtelee: A:ssa on 1000 tuotantokappaletta, B:ssä on 184 tuotantokappaletta ja C:ssä on 500 tuotantokappaletta.

Testauksessa jokainen paketti ajetaan neljä kertaa: kerran vanhalla optimoinnilla, kerran vanhalla optimoinnilla ja liukuvilla työalueilla, kerran uudella optimoinnilla sekä kerran uudella optimoinnilla ja liukuvilla työalueilla. Yhteensä kaksitoista eri ajoa, joista jokaisesta luodaan simulaattorilla raportit ja tallennetaan mittaustulokset.

Testauksessa huomioidaan myös liukuva työalue. Se ei sinällään kuulu tämän työn optimointiin, mutta se on olennainen osa simulaattoria ja tulevaisuuden linjaratkaisuja.

6.2 Tulosten tarkastelu

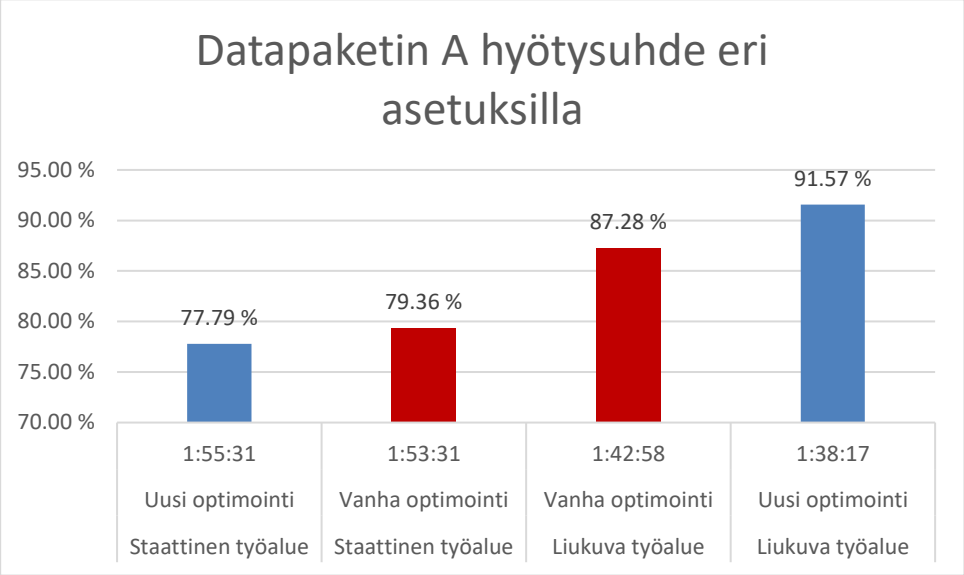
Testauksen lopputuloksena saatiin kahdentoista eri ajon raportit. Taulukossa 4 on yhteenveto työn kannalta tärkeimmistä arvoista. Siinä voidaan nähdä eri riveillä sijaitsevat ajot

ja niiden käytössä olevat datapaketit, työalueiden tyypit, käytetyn optimoinnin, suoritukseen kuluneen ajan (tunnit, minuutit ja sekunnit) sekä hyötysuhteen, joka prosentuaalisesti kuvastaa toteutuneen kapasiteetin suhdetta maksimikapasiteettiin. Tuloksista voidaan havaita huomattavat vaihtelut eri ajojen hyötysuhteissa. Suoritusaikojen välillä on myös suuria eroja, mutta ne osittain johtuvat käytetystä datapaketista: eri paketeissa on eri määrä tuotekappaleita ja tehtäviä. Datapakettien sisällä vaihtelut ovat maltillisempia, joskin silti havaittavissa.

Ajo	Datapaketti	Työalue	Optimointi	Suoritusaika	Hyötysuhde
1	A	Staattinen työalue	Vanha optimointi	1:53:31	79.36 %
2	A	Staattinen työalue	Uusi optimointi	1:55:31	77.79 %
3	A	Liukuva työalue	Vanha optimointi	1:42:58	87.28 %
4	A	Liukuva työalue	Uusi optimointi	1:38:17	91.57 %
5	B	Staattinen työalue	Vanha optimointi	0:22:13	76.81 %
6	B	Staattinen työalue	Uusi optimointi	0:22:32	75.60 %
7	B	Liukuva työalue	Vanha optimointi	0:20:15	85.36 %
8	B	Liukuva työalue	Uusi optimointi	0:19:19	90.29 %
9	C	Staattinen työalue	Vanha optimointi	0:59:04	76.81 %
10	C	Staattinen työalue	Uusi optimointi	0:57:23	79.12 %
11	C	Liukuva työalue	Vanha optimointi	0:53:34	85.00 %
12	C	Liukuva työalue	Uusi optimointi	0:50:19	90.47 %

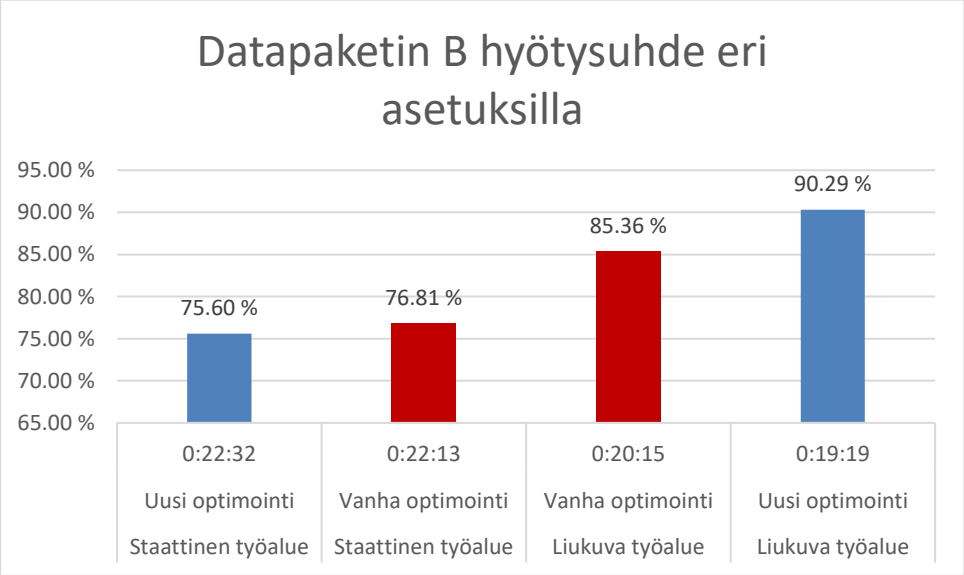
Taulukko 4. Testitulokset

Kuvassa 31 on kaavio datapaketin A hyötysuhteista eri asetuksilla. Siniset palkit kuvaavat uudella optimoinnilla ja punaiset palkit vanhalla optimoinnilla suoritettuja ajoja. Kaaviosta voidaan havaita, että uusi optimointi liukuvilla alueilla on hyötysuhteeltaan selkeästi paras kokonaisuus. Vanhaa optimointia käyttävien ajojen välillä ero on hurja: lähes 8 prosenttiyksikköä liukuvan työalueen eduksi. Staattisilla työalueilla uusi optimointi häviää vanhalle optimoinnille 1,5 prosenttiyksikköä.



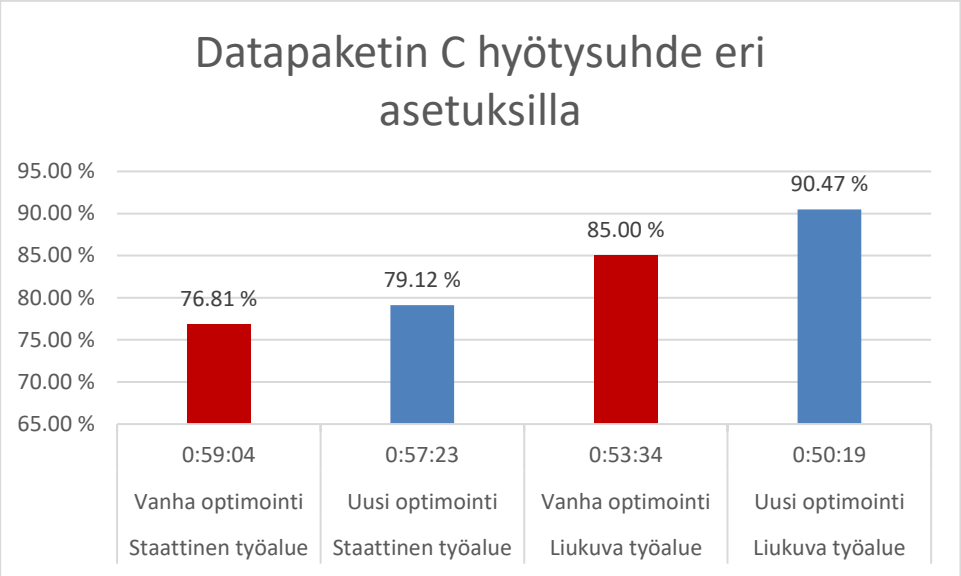
Kuva 31. Datapaketin A hyötysuhteet

Kuvassa 32 on kaavio datapaketin B hyötysuhteista eri asetuksilla. Tällä datapaketilla ajojen hyötysuhteet noudattavat täsmälleen samaa kaavaa kuin datapaketilla A: uusi optimointi on liukuvilla työalueilla paras ja staattisilla työalueilla huonoin. Ajojen eroissa on kuitenkin pientä heittoa: vanhaa optimointia käyttävien ajojen ero on vieläkin hurjempi, lähes 9 prosenttiyksikköä, kun taas staattisilla työalueilla uuden ja vanhan optimoinnin ero on hivenen pienempi, reilu 1 prosenttiyksikkö. Vaihtelut eroissa ovat kuitenkin verrattain melko pieniä ja johtuvat todennäköisesti datapaketti B:n huomattavasti pienemmästä otannasta: datapaketti B:ssä on 184 tuotantokappaletta, kun taas datapaketti A:ssa on 1000 tuotantokappaletta.



Kuva 32. Datapaketin B hyötysuhteet

Kuvassa 33 on kaavio datapaketin C hyötysuhteista eri asetuksilla. Tämän datapaketin ajot eroavat aikaisemmista: nyt uusi optimointi on selkeästi parempi sekä liukuvilla että staattisilla työalueilla. Staattisilla työalueilla uusi optimointi on vanhaa tehokkaampi lähes 2,5 prosenttiyksikköä.



Kuva 33. Datapaketin C hyötysuhteet

6.3 Pohdinta

Tuloksista voidaan havaita, että liukuvat työalueet tuovat jo yksinään huomattavan parannuksen tuotantolinjan hyötysuhteeseen. Testien tuloksissa liukuvien ja staattisten työalueiden välinen ero oli noin 10 %. Tämä on hyvin rohkaisevaa tulevia tuotantolinjaratkaisuja suunniteltaessa.

Uusi optimointi parantaa hyötysuhdetta huomattavasti liukuvilla työalueilla. Ero vanhan ja uuden optimoinnin välillä on noin 6 %. Ero uudella optimoinnilla liukuvalla työalueella ja vanhalla optimoinnilla staattisella työalueella on hurja: lähes 17 %. Uusi optimointi vaikuttaisi sopivan liukuville työalueille hyvin.

Staattisilla työalueilla vanha optimointi päihittää uuden optimoinnin kahdella datapaketilla kolmesta. Erot ovat kuitenkin suhteellisen pieniä: korkeintaan 2,5 prosenttiyksikköä. Uuden optimoinnin tappiot voidaan mahdollisesti selittää huonosti linjaan sopivilla optimointiparametreilla.

Tulosten perusteella voidaan havaita, että uusi optimointi ei ole automaattisesti vanhaa optimointia parempi tai huonompi. Linjalle sopivien parametrien etsiminen on hyvin hankalaa. Parametrit, jotka toimivat hyvin joillain linjalla, eivät välttämättä toimi toisilla ollenkaan. Tämä on havaittavissa näissäkin tuloksissa, kun uusi optimointi toimi huomattavasti paremmin liukuvilla työalueilla.

Tulokset ovat kuitenkin positiivisia. Liukuvat alueet tuovat huomattavan lisäyksen linjan hyötysuhteeseen ja uusi optimointi vaikuttaisi toimivan niillä erityisen hyvin. Vanhan optimoinnin päihittäminen staattisilla työalueilla on haastavaa, mutta ei kuitenkaan mahdotonta: olihan uusi optimointi yhdellä datapaketilla hivenen vanhaa nopeampi.

6.4 Kehityskohteet

Optimointikirjastoon lisättiin lukuisia parametreja, joilla pystytään vaikuttamaan optimoinnin lopputulokseen. Tulevaisuudessa olisi hyvä olla työkalu, jolla pystyttäisiin automaattisesti hakemaan tuotantolinjalle optimiparametrit. Siitä olisi ollut hyötyä työn tuloksia kerättäessä. Työkalussa voitaisiin käyttää hyödyksi oppivaa tekoälyä, joka osaisi tunnistaa tuotantolinjalla vallitsevan tilanteen ja toimia sen mukaan. Se olisi huomattava askel optimoinnin jatkokehitykselle.

Simulaattorin kehityksen yhteydessä luotiin konsepti, jossa optimointi olisi reaaliaikaista ja tehtäviä voitaisiin jakaa uudelleen tilanteen vaatiessa. Sitä ei tähän työhön kuitenkaan pystytty toteuttamaan, sillä se olisi vaatinut muutoksia tuotantolinjan muihin komponentteihin.

Yksi tehtävien jakamiseen suunniteltu parametri on tehtävän suoritusajan arviointi. On hyödyllistä tietää, kauanko robotilla kuluu aikaa jonkin tehtävän suorittamiseen, koska sillä voitaisiin saavuttaa tehtävien jakaminen ajan suhteen tasaisesti. Suoritusaika-arvion laskeminen ei kuitenkaan ole helppoa, sillä siihen tarvitaan tieto robotin tehtävän aikana suorittamista liikkeistä ja niihin kuluvista ajoista. Toinen vaihtoehto on perustaa arvio oikealla linjalla mitattuihin työskentelyaikoihin. Kumpaakaan vaihtoehtoa ei voitu soveltaa työn kehityksen aikana.

Halkeamien reitinhakuun löydettiin myös muita kehityskohteita kulkusuunnan optimoinnin lisäksi. Osa optimointiin tulevista tehtävistä on yhdistelmiä useammista halkeamista, jotka pitäisi suorittaa tietyssä järjestyksessä. Nämä tehtävät pitää optimoida erillään muusta

reitinhausta. Joissakin tehtävissä käytetään paikkausmateriaalia, joka saattaa suorituksen jälkeen levitä, jos robotti myöhemmin kulkee sen päältä. Reitinhakuun on luotava parametri, jolla voidaan määritellä, että kuljettava reitti ei voi kulkea suoritettujen tehtävien päältä.

7 Yhteenveto

Työn tavoitteena oli luoda uusi optimointikirjasto eräälle Rauten robotisoidulle tuotantolinjalle. Sen perustana toimi tuotantolinjasta aiemmin kehitetty simulaattori, jonka kehityksen aikana saatu kokemus toimi hyvänä lähtökohtana optimointimenetelmien suunnittelemiseen. Kokemuksen perusteella todettiin, että optimoinnin suurimpana haasteena on reitin optimointi, ja joka on käytännössä kauppamatkustajan ongelma. Tähän ongelmaan löydettiin sopivaksi ratkaisuksi Christofides-algoritmi, joka on suoritusajaltaan nopea, helppo implementoida ja tuottaa hyviä lopputuloksia.

Christofides-algoritmin toteutus sujui mutkattomasti. Kehitystä varten luotiin erillinen apuohjelma, mikä oli välttämätön työkalu algoritmin oikeellisuuden varmistamiseksi. Toteutuksen jälkeen algoritmi pystyttiin suoraan siirtämään optimointikirjastoon, jossa sitä pystyttiin testaamaan myös oikealla datalla. Testien perusteella havaittiin, että toteutus toimii, mutta se ei huomioi halkeamatehtävien kulkusuuntaa reitinhaussa, mikä ei tuota kaikissa tapauksissa parasta lopputulosta.

Käytännön toteutuksessa edettiin perinteisen ohjelmistonkehitysprosessin mukaan: työssä oli erilliset vaiheet vaatimusmäärittelylle, suunnittelulle, kehitykselle ja testaukselle. Kehityksessä käytettiin hyväksi Casey Muratorin [30] oppeja: aluksi luotiin toimiva käyttöliittymä, jonka ympärille lähdettiin rakentamaan mahdollisimman yksinkertaista toteutusta, jotta kirjasto saatiin nopeasti toimimaan muiden ohjelmistojen kanssa. Tämä mahdollisti nopean testaamisen aitoa tehdasympäristöä mukailevan virtuaaliympäristön kanssa.

Simulaattoriin tehtiin tuki optimointikirjastolle, ja se olikin suuressa osassa kaikissa kehityksen vaiheissa. Sen avulla pystyttiin tehokkaasti testaamaan kirjastoa eri asetuksilla, ja sen tarjoamat raportit helpottivat vertailua eri asetusten välillä. Raportteja ja vertailua hyödynnettiin myös työn tuloksissa. Tuloksissa havaittiin, että kehitetty optimointi on optimaalisilla asetuksilla vanhaa nopeampi, erityisesti liukuvilla työalueilla, mutta näiden asetusten löytäminen on haastavaa.

Lähteet

1 Tietoa Rautesta | Raute Oyj. Saatavilla: http://www.raute.fi/fi/tietoa-rautesta. Viitattu 12/11/2017, 2017.

2 Algorithm | Definition of Algorithm by Merriam-Webster. Saatavilla: https://www.merriam-webster.com/dictionary/algorithm. Viitattu 22/11/2017, 2017.

3 What is an Algorithm in Programming? - Definition, Examples & Analysis. Saatavilla: http://study.com/academy/lesson/what-is-an-algorithm-in-programming-definition-examples-analysis.html. Viitattu 22/11/2017, 2017.

4 Sedgewick R. Algorithms in C, Parts 1-4: Fundamentals, Data Structures, Sorting, Searching (3rd Edition). USA: Addison-Wesley; 1998.

5 Lawler E. L, Lenstra J. K, Rinnoy Kan A. H. G, Shmoys D. B. The Traveling Salesman Problem. Great Britain: John Wiley & Sons Ltd; 1986.

6 Kivinen J. 582456 Approksimointialgoritmit. Saatavilla: https://www.cs.helsinki.fi/u/jkivinen/opetus/appra/k10/luennot.pdf. Viitattu 29/11/2017, 2017.

7 Penton R. Data Structures for Game Programming. USA: Course PTR; 2003.

8 Pesonen M. E. Verkkoteorian alkeita. Saatavilla: http://cs.uef.fi/matematiikka/kurssit/MathematicsVisualizationMedia/CourseMaterial/VerkkoteoriaaSciFestiin2013.pdf. Viitattu 29/11/2017, 2017.

9 Schrijver A. On the History of the Shortest Path Problem. Saatavilla: https://www.math.uni-bielefeld.de/documenta/vol-ismp/32_schrijver-alexander-sp.pdf. Viitattu 29/11/2017, 2017.

10 LaMothe A. Inside Peliohjelmointi. Helsinki: Oy Edita Ab; 2000.

11 Christofides, N. Graph Theory. An Algorithmic Approach. USA: Academic Press; 1975.

12 Leonard Euler's Solution to the Konigsberg Bridge Problem | Mathematical Association of America. Saatavilla: https://www.maa.org/press/periodicals/convergence/leonard-eulers-solution-to-the-konigsberg-bridge-problem. Viitattu 22/11/2017, 2017.

13 Higgins P. M. Nets, Puzzles and Postmen. Great Britain: Oxford University Press; 2007.

14 KONIGSBERG PRIDGES PROBLEM. Saatavilla: https://physics.weber.edu/carroll/honors/konigsberg.htm. Viitattu 22/11/2017, 2017.

15 Fleury's Algorithm for printing Eulerian Path or Circuit – GeeksForGeeks. Saatavilla: http://www.geeksforgeeks.org/fleurys-algorithm-for-printing-eulerian-path/. Viitattu 22/11/2017, 2017.

16 Chinese Postman Problem. Saatavilla: http://www.utdallas.edu/~dzdu/cs6363/Chinese.htm. Viitattu 22/11/2017, 2017.

17 Graph Theory Techniques in Model-Based Testing. Saatavilla: http://www.oocities.org/model_based_testing/model-based.htm. Viitattu 22/11/2017, 2017.

18 How to solve the traveling salesman problem using dynamic programming. Saatavilla: https://www.quora.com/How-do-I-solve-the-traveling-salesman-problem-using-dynamic-programming. Viitattu 22/11/2017, 2017.

19 Kauppamatkustaja etsii lyhintä reittiä | Tiede. Saatavilla: https://www.tiede.fi/artikkeli/jutut/artikkelit/kauppamatkustaja_etsii_lyhinta_reittia. Viitattu 22/11/2017, 2017.

20 Christofides N. Worst-Case Analysis of a New Heuristic for the Travelling Salesman Problem. Pittsburgh, Pennsylvania: Camegie-Mellon University; February 1976.

21 Minimum spanning trees. Saatavilla: https://www.ics.uci.edu/~eppstein/161/960206.html. Viitattu 22/11/2017, 2017.

22 Kivinen J. 58131 Tietorakenteet Saatavilla: https://www.cs.helsinki.fi/u/jkivinen/opetus/tira/k08/kaikki.pdf. Viitattu 29/11/2017, 2017.

23 Lawler E. L. Combinatorial Optimization: Networks and Matroids. USA: Holt, Rinehart and Winston; 1976.

24 Cook W, Rohe A. Computing Minimum-Weight Perfect Matchings. Saatavilla: http://www.math.uwaterloo.ca/~bico/papers/match_ijoc.pdf. Viitattu 22/11/2017, 2017.

25 Perfect Matching -- from Wolfram MathWorld. Saatavilla: http://mathworld.wolfram.com/PerfectMatching.html. Viitattu 22/11/2017, 2017.

26 Gupta A. Advanced Algorithms. Saatavilla: http://www.cs.cmu.edu/~anupamg/advalgos15/lectures/lecture08.pdf. Viitattu 22/11/2017, 2017.

27 Euler and Hamiltonian Paths. Saatavilla: https://www.cs.sfu.ca/~ggbaker/zju/math/euler-ham.html#ham. Viitattu 22/11/2017, 2017.

28 Triangle Inequality – from Wolfram MathWorld.Saatavilla: http://mathworld.wolfram.com/TriangleInequality.html. Viitattu 29/11/2017, 2017.

29 The Traveling Salesman Problem and Heuristics. Saatavilla: https://ocw.mit.edu/courses/sloan-school-of-management/15-053-optimization-methods-in-management-science-spring-2013/lecture-notes/MIT15_053S13_lec17.pdf. Viitattu 29/11/2017, 2017.

30 Working on the Witness, Part 11. Saatavilla: https://mollyrocket.com/casey/stream_0019.html. Viitattu 12/11/2017, 2017.

31 Pohjonen R. Tietojärjestelmien kehittäminen. Vantaa: Docendo Finland Oy; 2002.

32 Testauksesta ja testaussuunnitelmasta. Saatavilla: http://www.sis.uta.fi/~tp54752/projektikurssit/2004_5/luennot/testaus.pdf. Viitattu 29/11/2017, 2017.