Aalto-yliopisto
Perustieteiden korkeakoulu
Teknillisen fysiikan ja matematiikan tutkinto-ohjelma

Vesa Husgafvel

Trimmitysongelman LP-relaksaation ratkaiseminen sarakkeita generoivalla algoritmilla ja brute-force-menetelmällä

Kandidaatintyö
Espoo 20.8.2012

Työn valvoja: Prof. Harri Ehtamo

Työn ohjaaja: DI Mirko Ruokokoski

Aalto-yliopisto
Perustieteiden korkeakoulu

Aalto-yliopisto
Perustieteiden korkeakoulu
Teknillisen fysiikan ja matematiikan tutkinto-ohjelma

KANDIDAATINTYÖN
TIIVISTELMÄ

Tekijä: Vesa Husgafvel

Työn nimi: Trimmitysongelman LP-relaksaation ratkaiseminen sarakkeita generoivalla algoritmilla ja brute-force-menetelmällä

Päivämäärä: 20.8.2012 **Kieli:** Suomi **Sivumäärä:** 3+27

Tutkinto-ohjelma: Teknillinen fysiikka ja matematiikka

Työn valvoja: Prof. Harri Ehtamo

Työn ohjaaja: DI Mirko Ruokokoski

Lineaaristen kokonaislukutehtävien joukolla on kasvava osuus matemaattisten optimointiongelmien parissa. Monet aikataulutus-, kuljetus- ja resurssienhallintaongelmat ovat esimerkkejä kokonaislukuoptimoinnin tehtävistä.

Tässä kandidaatintyössä tutkitaan trimmitysongelmaa (cutting stock problem), joka on eräs kustannusten minimointiin liittyvä kokonaislukuoptimoinnin tehtävä. Tämän ongelman ratkaisemiseksi esitellään simplex-menetelmän sekä branch-and-bound-algoritmin toimintaperiaatteet.

Numeerisessa osuudessa varsinaisen kokonaislukutehtävän ratkaisemisen sijasta tutkitaan LP-relaksaation, eli tehtävän jossa kokonaislukurajoitukset on poistettu, ratkaisemista. LP-relaksaation tarkastelu on perusteltua, sillä sen ratkaisu tarjoaa alarajan kokonaislukutehtävän ratkaisulle, mikä helpottaa alkuperäisen ongelman ratkaisemista. LP-relaksaation ratkaiseminen toteutetaan käyttäen sekä sarakkeita generoivaa algoritmia (column generation algorithm) että brute-force-menetelmää. Molemmat menetelmät esitellään trimmitysongelman käsittelyn yhteydessä.

Menetelmien tehokkuutta vertailtiin tutkimalla erilaisten instanssien ratkaisemiseen kulunutta laskenta-aikaa. Saatujen tulosten perusteella todettiin, että brute-force-menetelmä oli parempi pienen kokoluokan tehtävissä, kun taas sarakkeita generoiva algoritmi ratkaisi suuren kokoluokan tehtävät nopeammin. Sarakkeita generoiva algoritmi ratkaisi myös sellaiset tehtävät, joita brute-force ei kyennyt ratkaisemaan ennalta määrättyyn aikarajaan mennessä.

Avainsanat: lineaarinen ohjelmointi, sarakkeita generoiva algoritmi, brute-force-menetelmä, trimmitysongelma, simplex-menetelmä, branch-and-bound-algoritmi.

Sisältö

1 Johdanto

Tarkastellaan kauppamatkustajaa, jonka pitää vierailla tietyssä määrässä kaupunkeja ja palata lopulta lähtökaupunkiinsa, siten että kussakin kaupungissa tulee vierailtua täsmälleen kerran. Oletetaan lisäksi, että etäisyys kustakin kaupungista toiseen on tiedossa. Kysymys kuuluu, missä järjestyksessä kaupungit kannattaa vierailla, jotta kuljettu kokonaismatka olisi mahdollisimman lyhyt. Tämän *kauppamatkustajan ongelmana* (travelling salesman problem) tunnetun kokonaislukutehtävän ratkaiseminen on osoittautunut kuitenkin yllättävän vaikeaksi ongelman yksinkertaisesta kuvauksesta huolimatta: ei ole löydetty ratkaisualgoritmia, jonka laskenta-aika ei kasvaisi eksponentiaalisesti kaupunkien lukumäärän suhteen (Woeginger, 2003). Käytännössä joudutaan siis tyytymään ei-optimaalisiin ratkaisuihin.

Monissa kokonaislukutehtävissä, joiden ratkaiseminen on yhtä vaikeaa kuin kauppamatkustajan ongelman ratkaiseminen, päätösmuuttujien määrä voi olla niin suuri, ettei tehtävän käsitteleminen sellaisenaan ole mahdollista. Monesti näissä tehtävissä suurin osa päätösmuuttujista ei kuitenkaan vaikuta optimiratkaisuun. Tunnistamalla tällaisia muuttujia saadaan tehtävän kokoa rajattua ja näin ollen laskenta-aikaa parannettua.

Tässä kandidaatintyössä tarkastellaan *trimmitysongelmana* (cutting stock problem) tunnettua kokonaislukutehtävää, jossa tilanne on päätösmuuttujien osalta edellä kuvatunlainen. Tarkoituksena ei ole määrittää optimaalista kokonaislukuratkaisua vaan tarkastella tehtävän LP-relaksaatiota, eli tilannetta jossa sallitaan myös ei-kokonaislukuratkaisut. Tämä on mielekästä, sillä LP-relaksaation ratkaisu tarjoaa kokonaislukutehtävän ratkaisulle alarajan, mikä helpottaa alkuperäisen ongelman ratkaisemista.

Relaksaation ratkaisemista varten esitellään *sarakkeita generoiva algoritmi* (column generation algorithm), joka tarkastelee aluksi vain osaa päätösmuuttujista lisäten niiden määrää tehtävässä yksi kerrallaan, siten että tehtävän ratkaisu paranee koko ajan. Aina kun uusi päätösmuuttuja lisätään, kasvaa myös rajoitusmatriisin koko yhdellä sarakkeella, mistä seuraa menetelmän nimi. Algoritmi toteutetaan kokonaislukuoptimointiin erikoistuneella CPLEX-ohjelmistolla.

Toinen tapa relaksaation ratkaisemiseen on optimoida kohdefunktiota kaikkien päätösmuuttujien ja niiden määräämän rajoitusmatriisin suhteen. Tästä vaihtoehtoisesta ratkaisutavasta käytetään tässä työssä nimeä brute-force-menetelmä, ja se toteutetaan numeeriseen matriisilaskentaan perustuvalla MATLABilla.

Tämän kandidaatintyön aiheena on vertailla sarakkeita generoivan algoritmin ja brute-force-menetelmän ratkaisuaikoja trimmitysongelman relaksaation ratkaisemisessa, kun tehtävän kokoa muutetaan.

2 Lineaarisen ohjelmoinnin tehtävä

Lineaarisen kohdefunktion optimointia lineaaristen rajoitusehtojen suhteen kutsutaan lineaariseksi ohjelmoinniksi. Kyseinen aihealue on erittäin keskeinen optimoinnin osa-alue, sillä useat yritysmaailmaa kiinnostavat asiat kuten suunnittelu-, tuotanto- ja kuljetusongelmat voidaan mallintaa lineaarisella ohjelmoinnilla. Lisäksi jotkut kokonaislukuoptimoinnin algoritmit, kuten luvussa 5 esiteltävä branch-and-bound-algoritmi, ratkaisevat tehtävän sarjana lineaarisia tehtäviä.

Yleinen lineaarisen ohjelmoinnin minimointitehtävä on muotoa

$$\begin{aligned} z &= \min \quad \mathbf{f}'\mathbf{y} \\ &\quad \text{s.e.} \quad \mathbf{D}\mathbf{y} \geq \mathbf{b}. \end{aligned} \tag{2.1}$$

Sovellusten kannalta on tärkeää huomata, että muuttuja $\mathbf{y}$ voidaan esittää kahden ei-negatiivisen muuttujan avulla: $\mathbf{y} = \mathbf{y}^+ - \mathbf{y}^-$, missä $\mathbf{y}^+ \geq \mathbf{0}$ ja $\mathbf{y}^- \geq \mathbf{0}$. Epäyhtälörajoitus $\mathbf{D}\mathbf{y} \geq \mathbf{b}$ voidaan puolestaan esittää yhtälörajoituksena *ylijäämämuuttujan* $\mathbf{s}^- \geq 0$ avulla asettamalla $\mathbf{D}\mathbf{y} - \mathbf{s}^- = \mathbf{b}$. Jos epäyhtälörajoitus olisi muotoa $\mathbf{D}\mathbf{y} \leq \mathbf{b}$, puhuttaisiin *alijäämämuuttujasta* $\mathbf{s}^+ \geq 0$ ja merkittäisiin $\mathbf{D}\mathbf{y} + \mathbf{s}^+ = \mathbf{b}$.

Kun ei-negatiiviset muuttujat $\mathbf{y}^+$, $\mathbf{y}^-$ ja $\mathbf{s}^-$ kirjoitetaan pitkänä vektorina $\mathbf{x} = [\mathbf{y}^+; \mathbf{y}^-; \mathbf{s}^-]$, saadaan uusia muuttujia vastaavaksi kerroinvektoriksi $\mathbf{c} = [\mathbf{f}; -\mathbf{f}; \mathbf{0}]$ ja rajoitusmatriisiksi $\mathbf{A} = [\mathbf{D}, -\mathbf{D}, \mathbf{I}]$, missä $\mathbf{0}$ on nollavektori ja $\mathbf{I}$ yksikkömatriisi. Tehtävä voidaan tällöin kirjoittaa muotoon

$$\begin{aligned} z &= \min \quad \mathbf{c}'\mathbf{x} \\ &\quad \text{s.e.} \quad \mathbf{A}\mathbf{x} = \mathbf{b}, \\ &\qquad\quad \mathbf{x} \geq \mathbf{0}, \end{aligned} \tag{2.2}$$

mitä kutsutaan *lineaarisen ohjelmoinnin standardimuodoksi*. Muunnos on erittäin hyödyllinen, sillä useimmat algoritmit keskittyvät standardimuotoisen formulaation ratkaisemiseen. Oletetaan jatkon kannalta, että $\mathbf{A}$ on $m \times n$-matriisi, $\mathbf{c}$ ja $\mathbf{x}$ n-vektoreita sekä $\mathbf{b}$ m-vektori. Lisäksi vaaditaan, että $m \leq n$.

Lineaarisen ohjelmoinnin tehtävissä käypien vektoreiden joukko on aina samaa muotoa: käypä alue koostuu hypertasojen ja puoliavaruuksien äärellisen monesta leikkauksesta. Seuraava määritelmä havainnollistaa leikkausjoukkoa:

Määritelmä 2.1. $\mathbb{R}^n$:n osajoukko P on *monitahokas* (polyhedron), jos se on äärellisen monen puoliavaruuden leikkaus, eli

$$P = \{\mathbf{x} \in \mathbb{R}^n | \mathbf{A}\mathbf{x} \geq \mathbf{b}\}, \mathbf{A} \in \mathbb{R}^{m \times n}, \mathbf{b} \in \mathbb{R}^m.$$

Jos lisäksi pätee, että $\mathbf{x} \geq \mathbf{0}$ ja $\mathbf{A}\mathbf{x} = \mathbf{b}$, sanotaan monitahokasta standardimuotoiseksi.

Jatkossa oletetaan, että monitahokas sisältää vähintään kaksi pistettä, ja lisäksi käytetään seuraavaa notaatiota: $\mathbf{A}_j$ on matriisin j. sarake ja $\mathbf{a}_i'$ matriisin i. rivi.

Monitahokkaan ominaisuuksien tarkastelemiseksi otetaan esille *konveksin* joukon määritelmä:

Määritelmä 2.2. Joukko $S \subset \mathbb{R}^n$ on *konveksi*, jos $\forall \mathbf{x}, \mathbf{y} \in S$, ja $\forall \lambda \in [0,1]$, pätee, että $\lambda\mathbf{x} + (1-\lambda)\mathbf{y} \in S$.

Geometrisesti konveksisuus tarkoittaa sitä, että jos joukon kahden eri pisteen välille piirretään jana, niin silloin myös janan kaikki pisteet kuuluvat kyseiseen joukkoon. Osoittautuu, että lineaarisen kohdefunktion lokaali optimi on konveksissa joukossa myös globaali optimi (Shimizu 1997, 65), jolloin tarvitsee tutkia vain ratkaisun lähiympäristöä. Edellä olevan perusteella on syytä esittää

Lause 2.1. Jokainen monitahokas P on konveksi joukko.

Todistus. Osoitetaan aluksi, että konveksien joukkojen leikkaus on konveksi. Olkoon $\{S_i\}_{i \in I}$ kokoelma konvekseja joukkoja S_i, missä I on mielivaltainen indeksijoukko. Jos $\mathbf{x}, \mathbf{y} \in \bigcap_{i \in I} S_i \subset S_i$, niin määritelmän 2.2 perusteella $\lambda\mathbf{x} + (1-\lambda)\mathbf{y} \in S_i$, $\forall \lambda \in [0,1]$. Koska $\lambda\mathbf{x} + (1-\lambda)\mathbf{y} \in S_i \ \forall i \in I$, niin $\lambda\mathbf{x} + (1-\lambda)\mathbf{y} \in \bigcap_{i \in I} S_i$, mistä seuraa että $\bigcap_{i \in I} S_i$ on konveksi.

Oletetaan seuraavaksi, että $\mathbf{a}_i$ on monitahokkaan P rajoitusmatriisin $\mathbf{A}$ i. vaakavektori ja b_i vektorin $\mathbf{b}$ i. alkio. Jos nyt $\mathbf{x}$ ja $\mathbf{y}$ kuuluvat epäyhtälön $\mathbf{a}_i'\mathbf{x}$ määrittämään puoliavaruuteen, pätee että, $\mathbf{a}_i'\mathbf{x} \geq b_i$ ja $\mathbf{a}_i'\mathbf{y} \geq b_i$. Tällöin $\mathbf{a}_i'(\lambda\mathbf{x} + (1-\lambda)\mathbf{y}) \geq \lambda b_i + (1-\lambda)b_i = b_i \ \forall \lambda \in [0,1]$, mistä seuraa puoliavaruuden $\mathbf{a}_i'\mathbf{x} \geq b_i$ konveksisuus. Määritelmän 2.1 mukainen monitahokas voidaan kirjoittaa ekvivalentissa muodossa $P = \left\{ \mathbf{x} \in \mathbb{R}^n | \bigcap_{i=1}^m \{\mathbf{a}_i'\mathbf{x} \geq b_i\} \right\}$, mistä seuraa P:n konveksisuus. □

Monitahokkaan "kärkipisteiden"eli niiden pisteiden, jotka määräytyvät yksikäsitteisesti hypertasojen ja puoliavaruuksien leikkauksista, kutsutaan ääripisteiksi. Seuraava määritelmä antaa näille pisteille konstruktion:

Määritelmä 2.3. Vektori $\mathbf{x} \in P$ on monitahokkaan P *ääripiste*, jos ei löydy kahta $\mathbf{x}$:stä poikkeavaa vektoria $\mathbf{y}, \mathbf{z} \in P$ ja skalaaria $\lambda \in [0,1]$, siten että $\mathbf{x} = \lambda\mathbf{y} + (1-\lambda)\mathbf{z}$.

Tämän geometrisen määritelmän esittämistä voidaan pitää oleellisena, sillä ääripisteet ja LP-tehtävän *kantaratkaisut* osoittautuvat ekvivalenteiksi käsitteiksi (Bertsimas 1997, 50-52).

3 Simplex-menetelmä

Simplex-menetelmä on standardimuotoisen LP-tehtävän ratkaisemiseen kehitetty algoritmi, jonka George Bernard Dantzig esitti vuonna 1947 (Dantzig 1951). Algoritmin ideana on liikkua monitahokkaan ääripisteestä toiseen, kunnes optimaalinen ratkaisu saavutetaan.

3.1 Kantaratkaisut

Kiinnostuksen kohteena on aluksi löytää jokin ratkaisu tehtävälle (2.2), sillä oletuksella että $\mathbf{A}$:n rivit ovat lineaarisesti riippumattomat. Yleisesti aloitusratkaisun määrittäminen ei ole helppoa, mutta tämän työn aiheena olevalle trimmitysongelmalle (4.3), joka esitetään luvussa 4, käyvän ratkaisun löytäminen on kuitenkin triviaalia, minkä takia aiheen käsittely (Bertsimas 1997, 111-119) sivuutetaan.

Tapauksessa $m = n$ on olemassa $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$ muotoinen ratkaisu, koska $\mathbf{A}$ on ei-singulaarinen neliömatriisi. Yleisessä tapauksessa kun $n > m$, asetetaan $n - m$ kpl päätösmuuttujia nolliksi, siten että jäljelle jääviä muuttujia (m kpl) vastaavat sarakkeet ovat lineaarisesti riippumattomia. Lineaarisesti riippumattomien sarakkeiden joukkoa $\mathbf{B} = \left[\mathbf{A}_{B(1)}, \ldots, \mathbf{A}_{B(m)}\right]$ kutsutaan *kantamatriisiksi* (basis matrix), missä $\mathbf{A}_{B(i)}$ on matriisin i. sarake, $i = 1, \ldots, m$, sekä vastaavia päätösmuuttujia $\mathbf{x}_B = \mathbf{B}^{-1}\mathbf{b} = \left[x_{B(1)}, \ldots, x_{B(m)}\right]'$ *kantamuuttujiksi*. Jäljelle jäävät muuttujat ovat *ei-kantamuuttujia* (nonbasic variables) $\mathbf{x}_N$, (joille konstruktion perusteella $\mathbf{x}_N = \mathbf{0}$) missä N viittaa ei-kantamuuttujien indeksijoukkoon. Ei-kantamuuttujista ja kantamuuttujista $\mathbf{x}_B$ koostuvaa ratkaisua $\mathbf{x} = [\mathbf{x}_N; \mathbf{x}_B]$ kutsutaan *kantaratkaisuksi*. Jos lisäksi $\mathbf{x}_B \geq \mathbf{0}$, on kyseessä *käypä kantaratkaisu*.

Oletetaan, että standardimuotoiselle tehtävälle on löydetty kantamatriisi $\mathbf{B}$ sekä sitä vastaava käypä kantaratkaisu $\mathbf{x} \in P$. Koska P on monitahokkaana konveksi joukko (ja sisältää vähintään kaksi pistettä), on olemassa sellainen suunta, siten että pisteestä $\mathbf{x}$ kyseisen vektorin suuntaan kulkeminen ei välittömästi vie pois käyvästä alueesta:

Määritelmä 3.1. Olkoon $\mathbf{x}$ monitahokkaan P piste. Sanotaan, että vektori $\mathbf{d} \in \mathbb{R}^n$ on *käypä suunta* pisteessä $\mathbf{x}$, jos $\exists\ \theta > 0$, siten että $\mathbf{x} + \theta\mathbf{d} \in P$.

Tarkastellaan seuraavaksi tilannetta, jossa halutaan liikkua vektorin $\mathbf{x}$ komponentin x_j suuntaan θ:n verran, missä x_j on ei-kantamuuttuja. Toisin sanoen vaaditaan, että suunnalle $\mathbf{d}$ pätee $d_j = 1$ ja $d_i = 0$, kun $i \neq j$ ja $i, j \in N$. Merkitään, että kantamuuttujia $\mathbf{x}_B$ vastaava suuntavektori on $\mathbf{d}_B = [d_{B(1)}, d_{B(2)}, \ldots, d_{B(m)}]'$.

Jotta kantaratkaisusta $\mathbf{x}$ liikkuminen ei veisi ulos käyvästä alueesta, vaaditaan lisäksi, että $\mathbf{A}(\mathbf{x} + \theta\mathbf{d}) = \mathbf{b}$. Koska $\mathbf{Ax} = \mathbf{b}$ ja $\theta{>}0 \Rightarrow \mathbf{Ad} = \mathbf{0}$. Tällöin

$$\mathbf{0} = \mathbf{Ad} = \sum_{i=1}^{n} \mathbf{A}_i d_i = \sum_{i=1}^{m} \mathbf{A}_{B(i)} d_{B(i)} + \mathbf{A}_j = \mathbf{B}\mathbf{d}_B + \mathbf{A}_j.$$

Kantamatriisin ei-singulaarisuuden perusteella saadaan määrättyä kantavektoreita vastaavat suunnat

$$\mathbf{d}_B = -\mathbf{B}^{-1}\mathbf{A}_j. \tag{3.1}$$

Konstruktion tuloksena syntynyttä vektoria $\mathbf{d}$ nimitetään j :nneksi *kantavektoriksi.*

On huomattava, että edellä olevassa tarkastelussa ei käsitelty, toteutuuko kantamuuttujien ei-negatiivisuusrajoite uudessa pisteessä, eli päteekö $\mathbf{x}_B + \theta\mathbf{d}_B \geq 0$. Jos jollekin kantamuuttujalle $x_{B(i)} = 0$, voidaan joutua tilanteeseen, missä $\mathbf{d}$ ei olekaan käypä suunta, jolloin epäyhtälö toteutuu vain arvolla $\theta = 0$. Tämän tilanteen tarkasteluun palataan luvussa 3.4.

3.2 Redusoidut kustannukset

Tehtävän (2.2) ratkaisemisen kannalta on oleellista tietää, miten kohdefunktion arvo muuttuu siirryttäessä j:nnen kantavektorin $\mathbf{d}$ suuntaan. Yhden yksikön siirtymän jälkeen ($\theta = 1$) kohdefunktion arvon muutos on

$$\mathbf{c}'(\mathbf{x} + \mathbf{d}) - \mathbf{c}'\mathbf{x} = \mathbf{c}'\mathbf{d} = \mathbf{c}'_B\mathbf{d}_B + c_j = c_j - \mathbf{c}'_B\mathbf{B}^{-1}\mathbf{A}_j,$$

missä viimeinen yhtälö seuraa kaavasta (3.1). Tarkasteltavassa minimointitehtävässä (2.2) kohdefunktio $\mathbf{c}'\mathbf{x}$ vastaa usein hintaa, mikä joudutaan maksamaan, kun $\mathbf{x}$ kuvaa tuotetun hyödykkeen määrää. Siksi kertoimesta $\mathbf{c}$ käytetään myös nimitystä *kustannusvektori* (cost vector). Tällöin kohdefunktion arvon muutoksessa termi c_j voidaan tulkita yksikkökustannuksena muuttujan x_j kasvattamisesta ja termi $-\mathbf{c}'_B\mathbf{B}^{-1}\mathbf{A}_j$ kantamuuttujien aiheuttamana kustannuksena, siitä että yhtälön $\mathbf{Ax} = \mathbf{b}$ vaaditaan toteutuvan. On syytä määritellä redusoitujen kustannusten käsite.

Määritelmä 3.2. Olkoon $\mathbf{B}$ kantamatriisi ja $\mathbf{x}$ sitä vastaava kantaratkaisu sekä $\mathbf{c}_B$ kantamuuttujien kerroinvektori. Tällöin muuttujan x_j, $j = 1, \ldots, n$, *redusoitu kustannus* on

$$\overline{c}_j = c_j - \mathbf{c}'_B\mathbf{B}^{-1}\mathbf{A}_j.$$

Huomataan, että kantamuuttujille $\mathbf{x}_{B(i)}$ redusoitu kustannus $\overline{c}_{B(i)}$ on aina 0:

$$\overline{c}_{B(i)} = c_{B(i)} - \mathbf{c}'_B\mathbf{B}^{-1}\mathbf{A}_{B(i)} = c_{B(i)} - \mathbf{c}'_B\mathbf{e}_i = c_{B(i)} - c_{B(i)} = 0,$$

missä $\mathbf{e}_i$ on i:s yksikkövektori. Määritelmän 3.2 alustuksen perusteella kohdefunktion minimoinnin kannalta ei siis ole järkevää siirtyä j:nnen kantavektorin osoittamaan suuntaan, jos $\overline{c}_j \geq 0$. Kirjoitetaan havainto lauseeksi:

Lause 3.1. Olkoon $\mathbf{B}$ kantamatriisi, $\mathbf{x}$ sitä vastaava käypä kantaratkaisu ja $\overline{\mathbf{c}}$ redusoitujen kustannusten vektori kyseisessä pisteessä. Jos $\overline{\mathbf{c}} \geq \mathbf{0}$, niin $\mathbf{x}$ on optimaalinen.

Todistus. Olkoon $\overline{\mathbf{c}} \geq \mathbf{0}$ pisteessä $\mathbf{x}$ sekä $\mathbf{y}$ jokin käypä ratkaisu. Merkitään lisäksi, että $\mathbf{d} = \mathbf{y} - \mathbf{x}$. Käypyyden perusteella $\mathbf{Ax} = \mathbf{Ay} = \mathbf{b}$, jolloin

$$\mathbf{Ad} = \mathbf{Bd}_B + \sum_{i \in N} \mathbf{A}_i d_i = \mathbf{0},$$

missä N on ei-kantamuuttujia vastaava indeksijoukko. Ratkaisemalla $\mathbf{d}_B$ yllä olevasta yhtälöstä saadaan, että

$$\mathbf{d}_B = -\sum_{i \in N} \mathbf{B}^{-1}\mathbf{A}_i d_i$$

ja edelleen

$$\mathbf{c}'\mathbf{d} = \mathbf{c}'_B\mathbf{d}_B + \sum_{i \in N} c_i d_i = \sum_{i \in N}(c_i - \mathbf{c}'_B\mathbf{B}^{-1}\mathbf{A}_i)d_i = \sum_{i \in N} \overline{c}_i d_i.$$

Koska $x_i = 0\ \forall i \in N$ ja $y_i \geq 0\ \forall i \in \{1, 2, \ldots, n\}$, niin $d_i \geq 0\ \forall i \in N$. Oletuksen $\overline{\mathbf{c}} \geq \mathbf{0}$ perusteella nyt $\overline{c}_i d_i \geq 0\ \forall i \in N$, mistä seuraa, että $\mathbf{c}'\mathbf{d} \geq \mathbf{0}$. Koska $\mathbf{y}$ oli mielivaltainen käypä ratkaisu, on $\mathbf{x}$ optimaalinen. □

3.3 Uusi kantaratkaisu

Edellisessä luvussa todettiin, että käyvästä kantaratkaisusta $\mathbf{x}$ yksikkösiirtymä ($\theta = 1$) j:nnen kantavektorin $\mathbf{d}$ suuntaan muuttaa kohdefunktion arvoa määrällä $\overline{c}_j$. Vastaava muutos kun $\theta \geq 0$, on $\theta\overline{c}_j$. Lauseen (3.1) perusteella tiedetään, että siirtyminen vektorin $\mathbf{d}$ suuntaan on järkevää, vain jos $\overline{c}_j < 0$, jolloin kiinnostuksen kohteena on tehtävä

$$\theta^* = \max\{\theta \geq 0 | \mathbf{x} + \theta\mathbf{d} \in P\}. \tag{3.2}$$

Yhtälörajoitus $\mathbf{A}(\mathbf{x} + \theta\mathbf{d}) = \mathbf{b}$ toteutuu kantasuunnan $\mathbf{d}$ konstruktion perusteella, kun $\theta > 0$, ja tapauksessa $\theta = 0$ on toteutuminen triviaalia. Tarkasteltavaksi jäävät ei-negatiivisuusrajoitteet, joiden käsittely jaetaan kahteen osaan:

1. $\mathbf{d} \geq \mathbf{0}$, jolloin $\mathbf{x} + \theta\mathbf{d} \geq 0\ \forall\theta \geq 0$. Tällöin $\theta^* = \infty$, mistä seuraa, että $z = -\infty$.
2. $\exists i$ s.e. $d_i < 0$, jolloin $x_i + \theta d_i \geq 0 \Leftrightarrow \theta \leq -x_i/d_i$. Suurin mahdollinen θ^* saadaan nyt minimointitehtävän

$$\theta^* = \min_{\{i | d_i < 0\}} \Big(-\frac{x_i}{d_i} \Big) = \min_{\{i=1,\ldots,m | d_{B(i)} < 0\}} \Big(-\frac{x_{B(i)}}{d_{B(i)}} \Big) \tag{3.3}$$

ratkaisuna.

Olettaen että $\theta^* < \infty$ ja $l = \arg\min_i(-x_{B(i)}/d_{B(i)})$, jolloin $-x_{B(l)}/d_{B(l)} = \theta^* \Leftrightarrow x_{B(l)} + \theta^* d_{B(l)} = 0$. Olkoon uusi ratkaisu $\mathbf{y} = \mathbf{x} + \theta^*\mathbf{d}$, jolle pätee nyt $y_j = \theta^*$ ja $y_{B(l)} = 0$. Kun vanhan kantamatriisin $\mathbf{B}$ sarake $\mathbf{A}_{B(l)}$ korvataan sarakkeella $\mathbf{A}_j$, saadaan uusi matriisi

$$\overline{\mathbf{B}} = \left[\mathbf{A}_{B(1)}, \ldots, \mathbf{A}_{B(l-1)}, \mathbf{A}_j, \mathbf{A}_{B(l+1)}, \ldots, \mathbf{A}_{B(m)}\right]. \tag{3.4}$$

Seuraava lause takaa, että myös $\overline{\mathbf{B}}$ on kantamatriisi ja $\mathbf{y}$ sitä vastaava käypä kantaratkaisu.

Lause 3.2.

(a) $\overline{\mathbf{B}}$ on kantamatriisi

(b) $\mathbf{y} = \mathbf{x} + \theta^* \mathbf{d}$ on kantamatriisia $\overline{\mathbf{B}}$ vastaava käypä kantaratkaisu

Todistus. **(a)** Riittää näyttää, että $\overline{\mathbf{B}}$:n sarakkeet $\mathbf{A}_{\overline{B}(i)}$, $i = 1, \ldots, m$, ovat lineaarisesti riippumattomat. Tehdään vastaoletus, että sarakkeet $\mathbf{A}_{\overline{B}(i)}$, $i = 1, \ldots, m$, ovat lineaarisesti riippuvia, eli on olemassa kertoimet $\lambda_1, \ldots, \lambda_m$, joista vähintään yksi on nollasta poikkeava, siten että

$$\sum_{i=1}^{m} \lambda_i \mathbf{A}_{\overline{B}(i)} = \mathbf{0}.$$

Tästä seuraa myös vektoreiden $\mathbf{B}^{-1}\mathbf{A}_{\overline{B}(i)}$, $i = 1, \ldots, m$, lineaarinen riippuvuus

$$\sum_{i=1}^{m} \lambda_i \mathbf{B}^{-1} \mathbf{A}_{\overline{B}(i)} = \mathbf{0}.$$

Lisäksi on voimassa, että

$$\mathbf{B}^{-1}\mathbf{A}_{\overline{B}(i)} = \begin{cases} \mathbf{B}^{-1}\mathbf{B}_i = \mathbf{e}_i, & \text{kun } i \neq l, \\ \mathbf{B}^{-1}\mathbf{A}_j = -\mathbf{d}_B, & \text{kun } i = l, \end{cases}$$

missä $\mathbf{e}_i$ on i:s yksikkövektori. Yksikkövektorit ovat lineaarisesti riippumattomia, eli $\mathbf{e}_i(l) = 0 \ \forall i \neq l$, ja $\mathbf{d}_{B(l)} < 0$, jolloin myös vektorijoukko $\left\{\mathbf{B}^{-1}\mathbf{A}_{\overline{B}(i)}\right\}_{i=1}^{m}$ on lineaarisesti riippumaton. Saatiin ristiriita, jolloin alkuperäinen väite pätee.
(b) Pätee, että $\mathbf{y} \geq \mathbf{0}$, $\mathbf{A}\mathbf{y} = \mathbf{b}$, ja y_i=0, kun $i \notin \left\{\overline{B}(1), \ldots \overline{B}(m)\right\}$. Lisäksi $\overline{\mathbf{B}}$ on kantamatriisi kohdan **(a)** perusteella, jolloin $\mathbf{y}$ on määritelmän mukaan kyseistä matriisia vastaava käypä kantaratkaisu. □

3.4 Degeneroituvuus

Määritellään degeneroituvuuden käsite:

Määritelmä 3.3. Olkoon P standardimuotoinen monitahokas, $\mathbf{B}$ kantamatriisi ja $\mathbf{x}$ sitä vastaava käypä kantaratkaisu. Sanotaan, että kantaratkaisu on *degeneroitunut*, jos jonkin kantamuuttujan arvo on 0, eli $\exists i \in \{1, \ldots, m\}$, siten että $x_{B(i)} = 0$.

Jaetaan ei-negatiivisuusrajoitteen $\mathbf{x}_B + \theta^* \mathbf{d}_B \geq 0$ tarkastelu kahteen osaan:

1. Käypä kantaratkaisu $\mathbf{x}$ ei ole degeneroitunut. Tällöin $\mathbf{x}_B > \mathbf{0}$ ja $\exists\ \theta^* > 0$ siten, että $\mathbf{x}_B + \theta \mathbf{d}_B \geq \mathbf{0}$, eli uusi ratkaisu toteuttaa ei-negatiivisuusehdon.

2. Käypä kantaratkaisu $\mathbf{x}$ on degeneroitunut. Jos nyt $\exists l \in B$, siten että $x_{B(l)} = 0$ ja $d_{B(l)} < 0$, pätee, että $x_{B(l)} + \theta^* d_{B(l)} \geq 0$, jos ja vain jos $\theta^* = 0$. Vaikka uusi ratkaisu ei eroakaan vanhasta, voidaan kantamatriisia päivittää ($\mathbf{B} \hookrightarrow \overline{\mathbf{B}}$) asettamalla $\mathbf{A}_{B(l)} \hookrightarrow \mathbf{A}_j$ siinä toivossa, että uudesta kannasta päästään parempaan ratkaisuun. Kannanvaihdon seurauksena saatetaan myöhemmin kuitenkin päätyä takaisin vanhaan kantaan. Ilmiötä kutsutaan syklisyydeksi, joka voidaan estää sopivilla muuttujavalinnoilla. Valinta tehdään useimmiten *leksikografisen säännön* tai *Blandin säännön* perusteella (Bertsimas 1997, 108-111). Esitetään näistä jälkimmäinen:

Blandin sääntö

1. Valitaan j. kantasuunta, siten että $j = \min\{k \in N | \overline{c}_k < 0\}$.
2. Valitaan $x_{B(l)}$, siten että $l = \min\left\{i \in \{1, \ldots, m\} \mid -x_{B(i)}/d_{B(i)} = \theta^*, d_{B(i)} < 0\right\}$.

Säännön todistus (Chvátal 1983) sivuutetaan. Blandin sääntö estää syklisyyden, jolloin simplex-algoritmi saavuttaa optimaalisen ratkaisun äärellisessä ajassa.

Huomataan, että degeneroituneessa tilanteessa optimaalinen ratkaisu voidaan saavuttaa useammalla kantamatriisin valinnalla. Tästä johtuen on syytä määritellä vielä optimaalisen kantamatriisin käsite.

Määritelmä 3.4. Kantamatriisi $\mathbf{B}$ on *optimaalinen*, jos:

(a) $\mathbf{B}^{-1}\mathbf{b} \geq \mathbf{0}$, ja

(b) $\overline{\mathbf{c}} \geq \mathbf{0}$.

3.5 Simplex-iteraatio

Esitetään standardimuotoisen tehtävän simplex-iteraatio algoritmisena operaatiojonona perustuen lukujen 3.1 - 3.4 käsittelyyn:

1. Etsi kantamatriisi $\mathbf{B}$, jota vastaa käypä kantaratkaisu $\mathbf{x}$.
2. Määritä redusoidut kustannukset $\overline{c}_j$ kaikille ei-kantamuuttujille x_j, $j \in N$. Jos $\min_{j \in N} \overline{c}_j \geq 0$, on $\mathbf{x}$ optimaalinen ratkaisu: päätä algoritmi; muuten valitse j, niin että Blandin sääntö 1 toteutuu.
3. Määritä j:s kantasuunta $\mathbf{d}$. Jos $\mathbf{d} \geq 0$, tehtävän ratkaisu $z = -\infty$: päätä algoritmi. Muuten mene kohtaan 4.
4. Määritä $\theta^* = \min_{\left\{i=1,\ldots,m | d_{B(i)} < 0\right\}} \left(-\frac{x_{B(i)}}{d_{B(i)}} \right)$.
5. Muodosta uusi kanta $\overline{\mathbf{B}}$ asettamalla $\mathbf{A}_{B(l)} \hookrightarrow \mathbf{A}_j$, missä indeksi l on Blandin säännön 2 mukainen. Määritä uutta kantaa vastaava ratkaisu $\mathbf{y}$. Merkitse uutta ratkaisua alkuperäisillä muuttujilla: $\overline{\mathbf{B}} \hookrightarrow \mathbf{B}$ ja $\mathbf{y} \hookrightarrow \mathbf{x}$. Palaa kohtaan 2.

4 Trimmitysongelma

Teollisuudessa tuotantostrategiaansa suunnitteleva yritys haluaa maksimoida voittonsa: miten kysyntä voidaan tyydyttää pitämällä kustannukset mahdollisimman pieninä. Monesti kustannusten kannalta keskeisenä tekijänä on käytetyn raaka-aineen määrä. Raaka-aineen käytön minimointiin liittyvän *trimmitysongelman* (cutting stock problem) esitti ja muotoili ensimmäistä kertaa venäläinen taloustieteilijä Kantorovich vuonna 1939 etsiessään optimaalista tapaa sahata vaneria teollisuuden tarpeisiin (Kantorovich 1960).

Otetaan tarkastelun kohteeksi paperitehdas, joka tuottaa emorullaa, jonka pituus on W. Asiakasyritys i, $i = 1, 2, \ldots, m$, haluaa kuitenkin ostaa b_i kappaletta rullaa, jonka pituus on w_i. Oletetaan lisäksi, että $w_i, W \in \mathbb{Z}_+$ ja $w_i \leq W \; \forall i \in \{1, 2, \ldots, m\}$. Pienempiä rullia saadaan viipaloimalla emorullia erilaisilla leikkausmuoteilla. Kutakin leikkausmuottia $j = 1, 2, \ldots, n$ $(n \geq m)$, vastaa sarakevektori $\mathbf{A}_j$ tehtävän rajoitusmatriisissa $\mathbf{A}$, siten että matriisin alkio $a_{ij} \in \mathbb{Z}_+$ ilmaisee, kuinka monta kertaa rulla w_i esiintyy leikkausmuotissa j. Luonnollisesti yhdessä leikkausmuotissa olevien pienten rullien yhteispituus ei saa ylittää emorullan kokoa W, mistä saadaan rajoitusehdot

$$\sum_{i=1}^{m} a_{ij} w_i \leq W, \qquad j = 1, 2, \ldots, n. \tag{4.1}$$

Kysymys kuuluu, minkälaisia leikkausmuotteja (ja kuinka paljon niitä) paperitehtaan tulee käytää tyydyttäkseen asiakasyritysten kysynnät minimoiden samalla käytettyjen emorullien lukumäärän. Asetetaan

Pääongelma (IPM):

$$\begin{aligned} Z_{IP} \quad = \quad & \min \quad && \sum_{j=1}^{n} x_j \\ & \text{s.e.} \quad && \sum_{j=1}^{n} a_{ij} x_j \geq b_i, && i = 1, 2, \ldots, m, \\ & && x_j \in \mathbb{Z}_+, && j = 1, 2, \ldots, n, \end{aligned} \tag{4.2}$$

missä x_j on leikkausmuotin j mukaan leikattujen emorullien lukumäärä ja IPM= "integer programming master". Tässä työssä kiinnostuksen kohteena ei kuitenkaan ole kyseisen kokonaislukutehtävän ratkaiseminen, vaan tarkasteluissa rajoitutaan LP-relaksaatioon, missä muuttujien x_j kokonaislukurajoitukset on poistettu. Yleistä muotoa olevien kokonaislukutehtävien ratkaisemista käsitellään myöhemmin luvussa 5. Määritellään tehtävälle LP-relaksaatio eli

Lineaarisen ohjelmoinnin pääongelma (LPM):

$$\begin{aligned} Z_{LPM} \quad = \quad \min \quad & \sum_{j=1}^{n} x_j \\ \text{s.e.} \quad & \sum_{j=1}^{n} a_{ij} x_j \geq b_i, \qquad i = 1, 2, \ldots, m, \\ & x_j \geq 0, \qquad\qquad\quad j = 1, 2, \ldots, n, \end{aligned} \tag{4.3}$$

missä muuttujien x_j kokonaislukurajoitukset on poistettu.

4.1 Sarakkeita generoiva algoritmi

Sarakkeita generoiva algoritmi (column generation algorithm) on LP-tehtäville tarkoitettu menetelmä, joka ratkaisee toistuvasti osatehtävän, jossa rajoitusmatriisin kokoa kasvatetaan sarake kerrallaan, ja uusien sarakkeiden lisääminen perustuu redusoitujen kustannusten tarkasteluun. Sarakkeita generoivan algoritmin kehittivät 1960-luvulla Gilmore ja Gomory (1961, 1963) trimmitysongelman LP-relaksaation ratkaisemiseksi.

LPM-ongelman (4.3) ratkaisemista varten tarkastellaan tehtävää tapauksessa, missä rajoitusmatriisi $\mathbf{A}$ on korvattu $m \times k$, $m \leq k < n$, kokoisella matriisilla $\tilde{\mathbf{A}}_k \subset \mathbf{A}$. Tämän alitehtävän tarkastelu on perusteltua, sillä suurin osa LPM-ongelman päätösmuuttujista on ei-kantamuuttujia optimiratkaisussa. Halutaan siis löytää sellainen $\mathbf{A}$:n sarakkeiden ja päätösmuuttujien kombinaatio $(\tilde{\mathbf{A}}_k, \tilde{\mathbf{x}}_k)$, jolla rajoitetun ongelman kohdefunktio saavuttaa optimiarvon Z_{LPM}. Tämä toteutetaan kasvattamalla päätösmuuttujien $\tilde{\mathbf{x}}_k$ määrää sekä matriisin $\tilde{\mathbf{A}}_k$ kokoa sarake kerrallaan lähtien liikkeelle sopivasta alkuarvauksesta. Määritellään

Rajoitettu lineaarisen ohjelmoinnin pääongelma (RLPM):

$$\begin{aligned} Z_{RLPM} \quad = \quad \min \quad & \sum_{j=1}^{k} \tilde{x_j} \\ \text{s.e.} \quad & \sum_{j=1}^{k} \tilde{a_{ij}} \tilde{x_j} \geq b_i, \qquad i = 1, 2, \ldots, m, \\ & \tilde{x_j} \geq 0, \qquad\qquad\quad j = 1, 2, \ldots, k, \end{aligned} \tag{4.4}$$

missä $\forall i, j\ \exists l$, siten että $\tilde{a_{ij}} = a_{il}$, $l = 1, 2, \ldots, n$. Ratkaisukandidaatin $\tilde{\mathbf{x}}_k$ optimaalisuus voidaan tarkistaa laskemalla redusoidut kustannukset $\overline{c}_j$, missä $j \in \{1, \ldots, k\}$.

Olkoon $\mathbf{B} \subset \tilde{\mathbf{A}}_k$ RLPM-tehtävän optimaalinen, kokoa $m \times m$ oleva, kantamatriisi ja samalla siis jokin kantamatriisi LPM-tehtävälle, sillä $\tilde{\mathbf{A}}_k \subset \mathbf{A}$. Koska LPM-ongelman kohdefunktion kertoimille c_j=1 $\forall j \in \{1, \ldots, n\}$, voidaan muuttujan x_j redusoitu kustannus kirjoittaa muotoon

$$\overline{c}_j = 1 - \mathbf{p}'\mathbf{A}_j, \tag{4.5}$$

missä

$$\mathbf{p}' = \mathbf{c}_B'\mathbf{B}^{-1} = [1, \ldots, 1]'\mathbf{B}^{-1}. \tag{4.6}$$

Ratkaisu on optimaalinen, jos $\overline{\mathbf{c}} = [\overline{c}_1, \overline{c}_2, \ldots, \overline{c}_n]' \geq \mathbf{0}$, joten kiinnostuksen kohteena on löytää sellaiset sarakkeet, joille $\overline{c}_j < 0$. Sen sijaan, että jokainen sarake käytäisiin yksitellen läpi, voidaan ongelma muotoilla redusoitujen kustannusten minimointitehtävänä indeksin j suhteen.

Tämän minimointitehtävän, käypien leikkausmuottien ehdon (4.1) sekä alkioiden a_{ij} kokonaislukurajoituksen perusteella määritellään

Aliongelma:

$$\begin{aligned} \overline{c}_{min} \quad = \quad \min \quad & 1 - \sum_{i=1}^{m} p_i a_i \\ \text{s.e.} \quad & \sum_{i=1}^{m} w_i a_i \leq W, \\ & a_i \in \mathbb{Z}_+, \qquad\qquad i = 1, 2, \ldots, m, \end{aligned} \tag{4.7}$$

missä $a_i = a_{ij}$, jollakin $j \in \{1, \ldots, n\}$. Aliongelman ratkaisuna saatu sarake $\mathbf{A}_j = [a_1, \ldots, a_m]'$ lisätään RLPM-ongelman rajoitusmatriisiin ($\tilde{\mathbf{A}}_{k+1} \rightarrow \tilde{\mathbf{A}}_k$), lisätään päätösmuuttuja ($\tilde{\mathbf{x}}_{k+1} \rightarrow \tilde{\mathbf{x}}_k$) ja uusi RLPM ratkaistaan.

Rajoitusmatriisin $\tilde{\mathbf{A}}_k$ alkuarvauksena voidaan käyttää valintaa $\tilde{\mathbf{A}}_m = \mathbf{I}_m$, missä $\mathbf{I}_m$ on $m \times m$ yksikkömatriisi. Tällöin voidaan valita edelleen, että myös kantamatriisi $\mathbf{B} = \mathbf{I}_m$, jolloin $\overline{c}_j = 0 \; \forall j \in \{1, \ldots, m\}$, ja $\tilde{\mathbf{x}}_m = \mathbf{B}^{-1}\mathbf{b} = \mathbf{b} \geq \mathbf{0}$, eli kyseiselle RLPM-tehtävälle on löydetty optimaalinen kanta ja sitä vastaava ratkaisu. Käytännössä tämä ratkaisu vastaa sitä, että kustakin emorullasta W saadaan vain yksi pieni rulla w_i, $i = 1, \ldots, m$.

4.1.1 Aliongelman ratkaiseminen

Muotoillaan aliongelma ekvivalenttina maksimointiongelmana:

$$\begin{aligned} F(W) \quad = \quad \max \quad & \sum_{i=1}^{m} p_i a_i \\ \text{s.e.} \quad & \sum_{i=1}^{m} w_i a_i \leq W, \\ & a_i \in \mathbb{Z}_+, \qquad\qquad i = 1, 2, \ldots, m. \end{aligned} \tag{4.8}$$

Saatu tehtävä tunnetaan *kokonaislukuarvoisena selkärepun täyttöongelmana* (integer knapsack problem), jonka ratkaiseminen yleisessä tapauksessa ei ole helppoa. Jos tehtävän parametrit pysyvät rajoitettuina, voidaan ratkaiseminen toteuttaa tehokkaasti dynaamisen optimoinnin keinoin, eli menetelmällä jossa ratkaisualgoritmin toiminta perustuu rekursioon.

Tarkastellaan selkäreppuongelman alitehtävää

$$\begin{aligned} F(v) \quad = \quad & \max \quad && \sum_{i=1}^{m} p_i a_i \\ & \text{s.e.} \quad && \sum_{i=1}^{m} w_i a_i \leq v, \\ & && a_i \in \mathbb{Z}_+, \qquad\qquad i = 1, 2, \ldots, m, \end{aligned} \tag{4.9}$$

missä 'repun koko' W on korvattu v:llä ($v \leq W$). On luonnollista asettaa $F(v) = -\infty$ tapauksessa $v < 0$, sillä repun painon tulee aina olla ei-negatiivinen. Jos $0 \leq v < w_{min} = \min_i w_i$, täytyy päteä $F(v) = 0$, sillä repun painovaatimuksen täyttävää esinettä ei ole. Kun $v \geq w_{min}$, voidaan $F(v)$:n arvo määrätä täyttämällä reppu siten, että sen paino on korkeintaan $v - w_i$, ja sen jälkeen lisäämällä w_i:n painoinen esine. Esineen i valinta tulee suorittaa niin, että kohdefunktio saavuttaa maksimiarvonsa. Edellä esitetyn perusteella on saatu rekursiokaava

$$F(v) = \begin{cases} -\infty & \text{kun } v < 0, \\ 0 & \text{kun } 0 \leq v < w_{min}, \\ \max\limits_{i=1,\ldots,m} \{F(v - w_i) + p_i\} & \text{kun } v \geq w_{min}, \end{cases}$$

minkä avulla $F(v)$ voidaan laskea arvoilla $v \in \{w_{min}, w_{min} + 1, \ldots, W\}$. Viimeistä askelta $v = W$ ja optimaalista arvoa $F(W)$ vastaava esinejoukko $\mathbf{a}^*$ saadaan palautuvasti määrättyä pitämällä muistissa $F(v)$:n maksimoiva indeksi i kullakin v:n arvolla.

4.1.2 Sarakkeen generointi -iteraatio

Esitetään LPM-tehtävän (4.3) sarakkeen generointi -iteraatio operaatiojonona:

1. Muodosta RLPM-ongelma valitsemalla jokin $\tilde{\mathbf{A}}_k \subset \mathbf{A}$, siten että tehtävä on ratkeava.

2. Etsi RLPM-optimaalinen kantamatriisi $\mathbf{B}$ ja muodosta vektori $\mathbf{p}' = [1, \ldots, 1]'\mathbf{B}^{-1}$.

3. Etsi redusoitujen kustannusten minimi $\overline{c}_{min}$ ratkaisemalla aliongelma (4.7). Jos $\overline{c}_{min} \geq 0$, on kantamatriisi $\mathbf{B}$ LPM-optimaalinen: päätä algoritmi. Muuten mene kohtaan 4.

4. Muodosta $\tilde{\mathbf{A}}_{k+1}$ lisäämällä aliongelman ratkaisuna saatu sarake $\mathbf{A}_j$ rajoitusmatriisiin $\tilde{\mathbf{A}}_k$. Lisää $\tilde{\mathbf{x}}_{k+1}$ uuteen RLPM-tehtävään ja aseta $k + 1 \hookrightarrow k$. Palaa kohtaan 2.

4.2 Brute-force-menetelmä

Brute-force-menetelmässä rajoitusmatriisi $\mathbf{A}$ selvitetään kokonaisuudessaan etsimällä kaikki mahdolliset leikkausmuotit $\mathbf{A}_j$, jotka määräytyvät implisiittisesti rajoitusehdosta (4.1). Geometrisesti tehtävä vastaa tietyn alueen kaikkien käypien kokonaislukupisteiden määritystä. Käytännössä algoritmi testaa, kuuluuko tarkasteltava piste käypään joukkoon: jos kuuluu, niin lisätään kyseinen piste eli sarake rajoitusmatriisiin; jos ei, niin siirrytään tarkastelemaan toista pistettä. Koska menetelmä joutuu yksitellen käymään läpi hyvin suuren joukon pisteitä, käytetään ”brute-force” -nimeä.

Monitahokkaan sisältämien kokonaislukupisteiden määrän arviointi on epätriviaalia, ja useimmissa tapauksissa joudutaan tyytymään huonoihin approksimaatioihin. Aihetta on käsitelty laajemmin muun muassa teoksessa *Integer Points in Polyhedra* (Barvinok 2008). Tässä työssä leikkausmuottien määrää arvioidaan ylöspäin korvaamalla käypä alue m-ulotteisella suorakulmiolla. Leikkausmuottien rajoitusehdon määräämän hypertason ja i. koordinaattiakselin leikkauspiste a_i^* on

$$a_i^* = \frac{W}{w_i}, \tag{4.10}$$

jolloin i. akseli sisältää $\lfloor a_i^* \rfloor + 1$ kokonaislukupistettä välillä $[0, a_i^*]$. Tällöin kärkipisteiden $(\underbrace{a_1^*, 0, \ldots, 0}_{\text{m-kpl}}), (0, a_2^*, \ldots, 0), \ldots, (0, \ldots, a_m^*)$ määräämä suorakulmio sisältää kokonaislukupisteitä määrän

$$N = \prod_{i=1}^{m} \Big(\left\lfloor \frac{W}{w_i} \right\rfloor + 1 \Big). \tag{4.11}$$

Saatua approksimaatiota verrataan tulosten yhteydessä brute-force-menetelmällä määritettyjen pisteiden lukumäärään.

Brute-force-algoritmi toteutetaan MATLABilla käyttämällä silmukkarakennetta, joka kasvattaa kullakin kierroksella leikkausmuotissa $\mathbf{A}_j$ olevien rullien w_i lukumäärää aina yhdellä, kunnes muotti ei enää toteuta rajoitusehtoa (4.1). Kun rajoitusmatriisi $\mathbf{A}$ on tiedossa, ratkaistaan saadun IPM-tehtävän relaksaatio (eli LPM-tehtävä) hyödyntämällä MATLABin CPLEX-rajapintaa.

5 Branch-and-bound-algoritmi

Tämän luvun tarkoituksena on kuvata menetelmä, joka ratkaisee lineaarisen - yleistä muotoa olevan - kokonaislukutehtävän. Erityisesti, menetelmää voidaan soveltaa pääongelman (4.2) ratkaisemiseen, kun rajoitusmatriisi on määrätty. Menetelmä soveltuu myös aliongelman (4.8) ratkaisemiseen, vaikkakin luvussa 4.1.1 käsitelty algoritmi on kyseiselle tehtävälle tehokkaampi. Vaikka branch-and-bound -algoritmia ei tämän työn numeerisessa osuudessa käytetäkään, on mallin teoreettinen esittely nähty tarpeelliseksi.

Branch-and-bound on diskreetin optimoinnin algoritmi, joka systemaattisesti etsii lineaarisen ohjelmoinnin tehtävälle kokonaislukuratkaisuja rajaamalla hakualuetta kohdefunktion ylä- ja alarajatarkastelujen perusteella. Menetelmän esittelivät 1960-luvulla A. H. Land ja A. G. Doig (Land 1960).

Olkoon tarkastelun kohteena kokonaislukuarvoinen minimointitehtävä

$$\begin{aligned} z_{IP} \quad = \quad & \min \quad \mathbf{c}'\mathbf{x} \\ & \text{s.e.} \quad \mathbf{x} \in F, \\ & \qquad \mathbf{x} \in \mathbb{Z}_+^n, \end{aligned} \tag{5.1}$$

missä F koostuu yhtälö- ja epäyhtälörajoituksista. Muodostetaan ongelmalle kaksi alitehtävää jakamalla F kahteen pistevieraaseen osajoukkoon F_1 ja F_2, siten että $F \cap \mathbb{Z}_+^n = (F_1 \cup F_2) \cap \mathbb{Z}_+^n$ eli ositus säilyttää kaikki F:n kokonaislukupisteet. Kun osajoukot F_1 ja F_2 jaetaan edelleen pistevieraisiin osajoukkoihin F_3, F_4, F_5 ja F_6, siten että $F_1 \cap \mathbb{Z}_+^n = (F_3 \cup F_4) \cap \mathbb{Z}_+^n$ ja $F_2 \cap \mathbb{Z}_+^n = (F_4 \cup F_5) \cap \mathbb{Z}_+^n$, saadaan näin jatkamalla k-kpl pistevieraita alitehtäviä $IP(F_i), i \in \{1, 2, \ldots, k\}$, $k \in \mathbb{N}$. Tällöin IP-tehtävän optimiratkaisulle $\mathbf{x}^*$ pätee: $\exists i \in \{1, 2, \ldots, k\}$ s.e. $\mathbf{x}^* \in F_i$, eli optimi löydetään poimimalla paras ratkaisu alitehtävien ratkaisujoukosta. Alitehtävien optimiratkaisujen löytäminen voi olla yhtä vaikeaa kuin alkuperäisen ongelman ratkaiseminen, mutta alarajan löytäminen on helppoa. Kun tehtävälle IP(F_i), $i \in \{1 \ldots, k\}$, määritellään LP-relaksaatio LP(F_i):

$$\begin{aligned} b(F_i) \quad = \quad & \min \quad \mathbf{c}'\mathbf{x} \\ & \text{s.e.} \quad \mathbf{x} \in F_i, \\ & \qquad \mathbf{x} \geq \mathbf{0}, \end{aligned} \tag{5.2}$$

saadaan simplex-algoritmilla määritettyä tehtävän ratkaisu, joka toimii suoraan alitehtävän alarajana eli

$$b(F_i) \leq \min_{x \in F_i} \mathbf{c}'\mathbf{x}. \tag{5.3}$$

Mikä tahansa käypä kokonaislukuratkaisu tarjoaa puolestaan IP-tehtävälle ylärajan, joista pienintä löydettyä merkitään U:lla. Jos jollekin alitehtävälle pätee nyt $b(F_i) \geq U$, ei U:ta parempaa arvoa voida tässä haarassa enää saavuttaa, jolloin kyseisen haaran tutkiminen käy tarpeettomaksi. Tutkiminen voidaan lopettaa myös silloin, kun alitehtävä osoittautuu ratkeamattomaksi, tai löydetty ratkaisu on kokonaislukuarvoinen. Alitehtävät, joita ei ole vielä tutkittu, kuuluvat *aktiivisten alitehtävien joukkoon.*

Tapaan, jolla F saadaan jaettua haluttuihin osajoukkoihin, ei vielä ole otettu kantaa. Ideana on siis rajata käypää aluetta, siten että löydetty ei-kokonaislukuratkaisu $\mathbf{x}^*$ ei toteuta uusia rajoitusehtoja. Olkoon $x_j^* \notin \mathbb{Z}_+$ LP-relaksaation optimiratkaisun j. komponentti, jolloin jako

$$F_1 = F \cap \left\{x : x_j \leq \left\lfloor x_j^* \right\rfloor\right\}, F_2 = F \cap \left\{x : x_j \geq \left\lceil x_j^* \right\rceil\right\} \tag{5.4}$$

toteuttaa vaaditut ehdot. Alitehtävien osiinjako tapahtuu samalla periaatteella. Algoritmi säilyttää muistissa kaikki aktiiviset alitehtävät sekä parhainta löydettyä yläarajaa U vastaavan kokonaislukuratkaisun. Jos $U < \infty$ menetelmän pysähtyessä, on sitä vastaava ratkaisu optimaalinen. Erityisesti jos kohdefunktion kerroin $\mathbf{c}$ on kokonaislukuvektori, ja $\lfloor b(F) \rfloor = U$ jollakin U:n arvolla, on sitä vastaava ratkaisu optimaalinen. Jos taas $U = \infty$ algoritmin pysähtyessä, ongelmalla ei ole kokonaislukuratkaisuja.

Esimerkki 5.1. Tarkastellaan kokonaislukutehtävää

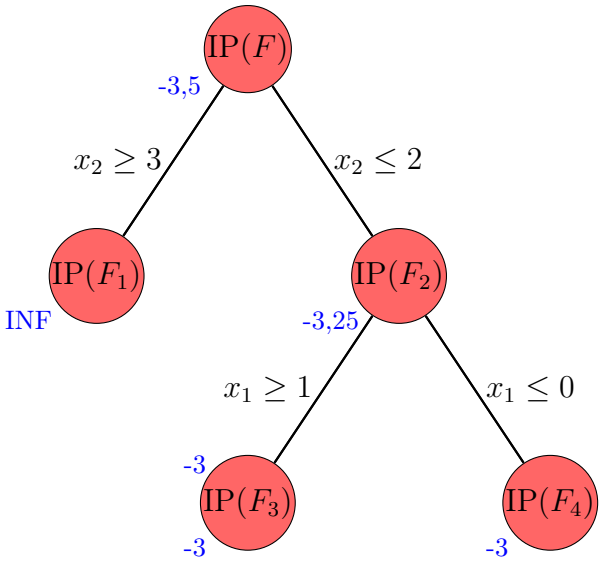
$$\begin{aligned} z_{IP} \quad = \quad & \min \quad x_1 - 2x_2 \\ & \text{s.e.} -4x_1 + 6x_2 \leq 9, \\ & \qquad x_1 + \ x_2 \leq 4, \\ & \qquad x_1, x_2 \in \mathbb{Z}_+. \end{aligned}$$

Alussa tehtävälle ei tiedetä mitään ylärajaa, joten asetetaan, että $U = \infty$, ja merkitään rajoitusten muodostamaa käypää aluetta F:llä. Tehtävän LP-relaksaation $\mathrm{LP}(F)$ optimiratkaisuksi saadaan $\mathbf{x} = (\frac{3}{2}, \frac{5}{2})$ ja kohdefunktion arvoksi $b(F) = -3,5$. Muodostetaan alitehtävät $\mathrm{IP}(F_1)$ ja $\mathrm{IP}(F_2)$, missä $F_1 = \{\mathbf{x} \in F | x_2 \geq 3\}$ ja $F_2 = \{\mathbf{x} \in F | x_2 \leq 2\}$. Aktiivisten alitehtävien joukko on nyt $\{\mathrm{IP}(F_1), \mathrm{IP}(F_2)\}$, joista $\mathrm{IP}(F_1)$ voidaan poistaa, kun huomataan, että $\mathrm{LP}(F_1)$ on ratkeamaton, eli $b(F_1) = \infty$. $\mathrm{LP}(F_2)$:n ratkaisuksi saadaan $\mathbf{x}^2 = (\frac{3}{4}, 2)$ ja $b(F_2) = -3,25$. Jaetaan nyt $\mathrm{IP}(F_2)$ kahteen alitehtävään $\mathrm{IP}(F_3)$ ja $\mathrm{IP}(F_3)$, missä $F_3 = \{\mathbf{x} \in F_2 | x_1 \geq 1\}$, ja $F_4 = \{\mathbf{x} \in F_2 | x_1 \leq 0\}$. Aktiivisten alitehtävien joukko on nyt $\{\mathrm{IP}(F_3), \mathrm{IP}(F_4)\}$. Relaksaation $\mathrm{LP}(F_3)$ ratkaisuksi saadaan $\mathbf{x}^3 = (1, 2)$ ja $b(F_3) = -3$. Koska $\mathbf{x}^3 \in \mathbb{Z}_+^2$, ollaan löydetty uusi yläraja $U = -3$ ja $\mathrm{IP}(F_3)$ voidaan poistaa aktiivisten tehtävien joukosta. Alitehtävän $\mathrm{IP}(F_4)$ relaksaation ratkaisuksi saadan $\mathbf{x}^4 = (0, \frac{3}{2})$ ja $b(F_4) = -3$. Koska $b(F_4) \geq U$ ja aktiivisten alitehtävien joukko on tyhjä, on alkuperäisen tehtävän optimiratkaisu $\mathbf{x}^3 = (1, 2)$ ja vastaava kohdefunktion arvo $z_{IP} = -3$. Tehtävän ratkaisua on havainnollistettu kuvan 1 hakupuussa.

Relaksaation $\mathrm{LP}(F_4)$ ratkaisun määrittäminen ei itse asiassa olisi ollut tarpeellista, sillä $\lfloor b(F) \rfloor = -3 = b(F_3) = U$, mikä jo takaa ratkaisun $\mathbf{x}^3$ optimaalisuuden.

Ei-kokonaislukuarvoisia muuttujia x_j^*, joiden suhteen haarautuminen voidaan toteuttaa, on useimmiten monia, jolloin tarvitaan jokin valintakriteeri. Yleinen käytössä oleva keino on valita sellainen muuttuja $x_{j'}^*$, jonka arvo on kauimpana kokonaisluvusta:

$$j' = \arg \max_{j=1,\ldots,n} \min \left\{x_j^* - \left\lfloor x_j^* \right\rfloor, 1 - (x_j^* - \left\lfloor x_j^* \right\rfloor)\right\}. \tag{5.5}$$



Kuva 1: Esimerkin 5.1 branch-and-bound hakupuu, missä $b(F) = -3,5$, $b(F_1) = \infty$, $b(F_2) = -3,25$, $b(F_3) = U = -3$ ja $b(F_4) = -3$.

Myös tapa, jolla aktiivisten alitehtävien valinta suoritetaan, vaikuttaa algoritmin suoritusaikaan. Valintametodina voidaan käyttää esimerkiksi *syvyyshakua* (depth-first search), eli menetelmää jossa kukin haara tutkitaan niin pitkälle kuin mahdollista, ennen kuin siirrytään seuraavan haaran käsittelyyn. Toinen vaihtoehto on käyttää *parhaan solmun strategiaa* (best-bound search), jossa pienin löydetty alaraja $\min_i b(F_i)$ määrää alitehtävän valinnan. Parhaan metodin valinta on usein vaikeaa ja vaihtelee tapauskohtaisesti.

5.1 Branch-and-bound-iteraatio

Branch-and-bound-algoritmin toimintaa kuvaa seuraava operaatiojono:

1. Aseta $U = \infty$. Ratkaise IP(F):n relaksaatio LP(F). Jos ratkaisu $\mathbf{x}^* \in \mathbb{Z}_+^n$, kyseessä IP-optimi: päätä algoritmi. Jos LP(F) ratkeamaton, ei ratkaisua: päätä algoritmi. Muulloin pidä muistissa $b(F)$.

2. Valitse muuttuja $x_j^* \notin \mathbb{Z}_+$, jonka suhteen haarautuminen tehdään ja muodosta aktiiviset alitehtävät.

3. Jos aktiivisten alitehtävien joukko on tyhjä, on U:ta vastaava ratkaisu optimaalinen: päätä algoritmi. Muulloin valitse aktiivinen alitehtävä IP(F_i) ja ratkaise LP(F_i).

4. Jos LP(F_i) on ratkeamaton, eliminoi haara ja palaa kohtaan 3.

5. Jos $b(F_i) < U$ ja $\mathbf{x}^* \notin \mathbb{Z}_+^n$, palaa kohtaan 2.

6. Jos $b(F_i) < U$ ja $\mathbf{x}^* \in \mathbb{Z}_+^n$ sekä $\lfloor b(F) \rfloor = b(F_i)$, kyseessä IP-optimi: päätä algoritmi. Muulloin aseta $U = b(F_i)$ ja pidä ratkaisu muistissa. Eliminoi haara ja palaa kohtaan 3.

7. Jos $b(F_i) \geq U$, eliminoi haara ja palaa kohtaan 3.

6 Tulokset

Sarakkeita generoivan algoritmin ja brute-force-menetelmän laskenta-aikoja LPM-instanssien (4.3) ratkaisemisessa vertailtiin muuttamalla rullien lukumäärää m ja emorullan kokoa W, kun rullien kokoja w_i ja vastaavia kysyntöjä b_i, $i = 1, \ldots, m$ pidettiin vakioina. Vaikka m ja W ovat myös LPM-tehtävän parametreja, puhutaan niistä selkeyden vuoksi muuttujina. Näin vältytään sekoittamasta suureita, mitä ollaan muuttamassa, ja mitä pidetään vakioina.

6.1 Parametrien valinta

Rullakoko- w_i ja kysyntäparametreina b_i, $i = 1, \ldots, m$, käytettiin MATLABin satunnaislukugeneraattorin `randi` arpomia kokonaislukuja, missä luvut tuotettiin väleiltä $[l_w, u_w] = [100, 500]$, ja $[l_b, u_b] = [0, 100]$. Parametrien arvoja ei generoitu uudestaan, jos emorullan kokoa muutettiin pitämällä lukumäärää m vakiona. Kun lukumäärää m muutettiin, generoitiin myös parametrit uudestaan.

Kun rullien lukumäärä $m \in \{10, 20, \ldots, 100\}$, ja emorullan koko $W \in \{500, 600, 700, 800, 900\}$, saatiin tutkittavien menetelmien laskenta-ajoiksi LPM-instansseissa taulukon 1 mukaiset tulokset. Taulukko 1 on esitetty liitteessä A. Lisäksi liitteen B taulukossa 2 on esitetty sarakkeita generoivan algoritmin käyttämien leikkausmuottien lukumäärä, brute-forcen muodostaman rajoitusmatriisin koko sekä kaavan (4.11) mukainen teoreettinen yläraja N käsitellyissä LPM-instansseissa. Taulukoissa on käytetty notaatioita SG=”sarakkeen generointi” ja BF=”brute-force”.

Tässä vaiheessa on syytä mainita, että muuttujien m ja W arvojoukkojen $\{10, 20, \ldots, 100\}$ ja $\{500, 600, 700, 800, 900\}$ sekä parametrien w_i ja b_i vaihteluvälien valinta ei perustunut minkään todellisen trimmitysongelman käsittelyyn. Arvojoukot pyrittiin lähinnä valitsemaan sellaisiksi, että instanssien ratkaisemiseen tarvittava laskenta-aika riippuisi merkittävästi muuttujien m ja W arvoista, mutta pysyisi silti kohtuullisena. Parametrien vaihteluvälien valinnalla haluttiin ennen kaikkea taata laskenta-ajan kohtuullisuus.

6.2 Brute-force-menetelmän laskenta-ajat

6.2.1 Muuttujien m ja W vaikutus laskenta-aikaan

Muuttujan m päävaikutusta tehtävän ratkaisuaikaan oli vaikea tutkia, sillä rullakoko- w_i ja kysyntäparametrit b_i, $i = 1, \ldots, m$, generoitiin uudestaan, kun m:n arvoa muutettiin. Periaatteessa muuttujan m päävaikutuksen sijasta pitäisikin puhua kyseisen muuttujan sekä parametrien w_i ja b_i yhteisvaikutuksesta. Jatkossa kuitenkin oletetaan, että parametrien vaikutus ei ole yhtä voimakasta kuin muuttujan m arvon vaikutus. Oletuksen toteutumista tarkastellaan tarkemmin kohdassa 6.2.2. Muuttujan W päävaikutuksen suhteen samaa ongelmaa ei sen sijaan esiinny, koska parametrit pysyivät vakioina W:n muuttuessa.

Taulukon 1 mukaan brute-force-menetelmän laskenta-aika kasvoi erittäin voimakkaasti muuttujien m ja W funktiona. Tulos on selitettävissä taulukon 2 arvojen

perusteella: muuttujien m ja W kasvaessa käypien leikkausmuottien määrä lisääntyi myös erittäin nopeasti, jolloin MATLABin vaatiman muistin tarve kasvoi myös voimakkaasti, mikä näkyi suoraan laskenta-ajassa. Jos lisäksi tutkitaan muuttujien m ja W arvon vaikutusta laskenta-aikaan, huomataan, että W:n arvon muuttamisella oli keskimäärin suurempi vaikutus ratkaisuaikaan kuin muuttujan m arvon muuttamisella.

Vertailemalla tarkasteltavien menetelmien laskenta-aikoja todetaan että, brute-force-menetelmä oli sarakkeita generoivaa algoritmia nopeampi, kun emorullan koko W ja rullien lukumäärä m pysyivät riittävän pieninä. Useimmiten tämä vastasi tilannetta, jossa $m \leq 40$, tai $W \leq 600$. Lisäksi menetelmän paremmuus suhteessa sarakkeita generoivaan algoritmiin laski nopeasti liikuttaessa alueella, jossa $m \geq 40$, ja $W \geq 700$. Taulukossa 1 asiaa on havainnollistettu merkitsemällä sinisellä värillä sellaisia tapauksia, joissa brute-forcen laskenta-aika oli 45-100 % sarakkeita generoivan algoritmin laskenta-ajasta ja punaisella värillä tapauksia, joissa sarakkeita generoiva algoritmi oli nopeampi.

Jos leikkausmuottien lukumäärien ylärajoja N verrataan todellisiin lukumääriin, voidaan todeta, että arvio ei kerro kovinkaan paljon oikeista arvoista. Approksimaatiokaavasta käy kuitenkin ilmi se tosiasia, että käypien leikkausmuottien määrä on sitä suurempi, toisaalta mitä isompi on W ja toisaalta mitä pienempi on w_i, $i = 1, \ldots, m$. LPM-instanssien laskenta-aikojen edes likimääräinen ennustaminen olisi silti ollut vaikeaa, sillä leikkausmuottien lukumäärän ja tehtävän koon välistä tarkkaa riippuvuussuhdetta ei kyetty määrittämään.

Vaikka taulukossa 1 ei ole eriteltynä, kuinka kauan brute-force-menetelmällä meni rajoitusmatriisin muodostamiseen ja sen jälkeen tehtävän ratkaisemiseen simplex-algoritmilla, on mainittavan arvoista todeta, että jälkimmäinen oli prosessina huomattavasti nopeampi. Osasyynä tähän on se, että MATLABin kutsuma CPLEX osaa esikäsitellä tehtävää, niin että ratkaistava tehtävä helpottuu. Esikäsittely poistaa esimerkiksi lineaarisesti riippuvia rivi- ja sarakevektoreita, mikä pienentää tehtävän kokoa. Leikkausmuottien etsintävaiheessa mitään tällaista esikäsittelyä ei tapahtunut. Rajoitusmatriisin muodostaminen oli siis työläin vaihe brute-force-menetelmässä.

6.2.2 Parametrien vaikutus laskenta-aikaan

Parametrien vaikutus brute-force-menetelmän laskenta-aikaan pystyttiin erottelemaan muuttujan m ja parametrien yhteisvaikutuksesta, kun instanssien ($m = 90$, $500 \leq W \leq 700$) ja ($m = 100$, $500 \leq W \leq 700$) ratkaisuaikoja verrattiin toisiinsa. Kun $m = 90$, oli ratkaiseminen hitaampaa, kuin tapauksessa $m = 100$, vaikka tehtävän leikkausmuottien lukumäärän olisi odotusarvoisesti pitänyt lisääntyä dimension m kasvaessa. Parametrien vaikutuksesta johtuen leikkausmuottien lukumäärät olivat kuitenkin pienemmät tapauksissa, joissa $m = 100$. On huomattavaa, että kyseinen laskenta-ajan pieneneminen parametrien vaikutuksesta vie pohjaa oletukselta, että muuttujan m vaikutus olisi dominoivaa parametrien vaikutukseen verrattuna.

Jos tarkastellaan, kuinka monta prosenttia laskenta-aika kasvoi, kun muuttujan m arvoa kasvatettiin 10:llä, huomataan, että kasvu oli poikkeuksellisen suurta verrattaessa instansseja ($m = 30$, $700 \leq W \leq 900$) ja ($m = 40$, $700 \leq W \leq 900$)

toisiinsa. Esimerkiksi laskenta-aika oli tapauksessa ($m = 40$, $W = 900$) yli 22 000 % suurempi kuin tapauksessa ($m = 30$, $W = 900$). Myös tapauksissa ($m = 40$, $W = 700$) ja ($m = 40$, $W = 800$) kasvua tapahtui yli 2000 % verrattuna tapauksiin ($m = 30$, $W = 700$) ja ($m = 30$, $W = 800$). Laskenta-ajan kasvu oli näissä instanssseissa niin suurta, että on oletettavaa, että parametreilla b_i ja w_i oli (muuttujan m ohella) merkittävää vaikutusta kyseisiin tuloksiin.

6.2.3 Poikkeavat havainnot

Luvussa 6.2.1 todettiin, että brute-force-menetelmän laskenta-ajan ja leikkausmuottien lukumäärän välillä vallitsee voimakas positiivinen riippuvuus. Jos tarkastellaan taulukon 1 ratkaisuaikoja tapauksissa $m = 10$ ja $W \in \{500, 600, 700\}$, huomataan, että brute-force-menetelmän laskenta-aika lyheni emorullan koon kasvaessa ja taulukon 2 mukaan käypien leikkausmuottien lukumäärä kasvoi samalla. Aiemmin havaittu riippuvuussuhde ei siis tunnu pätevän. Tulos on kuitenkin selitettävissä sillä, että ajon aikana MATLAB joutuu kääntämään kirjoitetun koodin toiseen muotoon. Muistin takia kääntäminen on nopeampaa, kun sama koodi käännetään uudestaan, mikä selittää havaitun epäloogisuuden.

6.3 Sarakkeita generoivan algoritmin laskenta-ajat

6.3.1 Muuttujien m ja W vaikutus laskenta-aikaan

Sarakkeita generoivan algoritmin laskenta-aika riippui voimakkaasti muuttujien m ja W arvoista - brute-force-menetelmän tavoin. Tulos ei ole yllättävä, sillä muuttujan m arvo vaikutti sekä aliongelman (4.7) että RLPM-tehtävän ratkaisunopeuteen. Aliongelman ratkaisemiseksi tarvittavien laskuoperaatioiden määrä riippui lisäksi tekijästä W. Lisäksi muuttujalla W havaittiin keskimäärin olevan suurempi vaikutus laskenta-aikaan kuin muuttujalla m.

Sarakkeita generoivan algoritmin ratkaisuajan havaittiin kasvavan käytettyjen leikkausmuottien lukumäärän mukaan. Tämäkään tulos ei ole yllättävä, sillä RLPM-tehtävän ratkaiseminen on sitä raskaampaa, mitä suuremmaksi rajoitusmatriisi $\tilde{\mathbf{A}}_k$ on kasvanut. Vaikka CPLEX esikäsitteleekin RLPM-tehtävää ennen sen ratkaisemista, kasvaa menetelmän kompleksisuus silti voimakkaasti generoitujen sarakkeiden lukumäärän funktiona. Kompleksisuuden kasvu näkyi korkeina laskenta-aikoina (yli 8 min) tehtävissä, joissa käytettyjen sarakkeiden lukumäärä oli luokkaa 250 tai sitä suurempi.

Taulukon 1 mukaan sarakkeita generoiva algoritmi ratkaisi isot tehtävät nopeammin kuin brute-force-menetelmä sekä sellaiset tehtävät, joita brute-force ei kyennyt ratkaisemaan 5 tunnin aikarajaan mennessä. Lähinnä näitä tilanteita vastasivat muuttujien arvot $m \geq 40$, ja $W \geq 800$. Lisäksi sarakkeita generoivan algoritmin tehokkuus suhteessa brute-force-menetelmään parani nopeasti tällä alueella, kun muuttujien arvoja kasvatettiin. Algoritmin käyttämien leikkausmuottien lukumäärä sekä instanssien ratkaisuajat eivät myöskään kasvaneet yhtä voimakkaasti muuttujien m ja W suhteen kuin brute-force-menetelmällä. Algoritmien tehokkuusvertailua on havainnollistettu taulukossa 1 merkitsemällä sinisellä värillä sellaisia tapauk-

sia, joissa brute-forcen laskenta-aika oli 45-100 % sarakkeita generoivan algoritmin laskenta-ajasta ja punaisella värillä tapauksia, joissa sarakkeita generoiva algoritmi oli nopeampi.

6.3.2 Parametrien vaikutus laskenta-aikaan

Sarakkeita generoivan algoritmin laskenta-aika pienentyi muutamassa tapauksessa, kun muuttujan m arvoa kasvatettiin. Esimerkiksi instanssin ($m = 30$, $W = 700$) ratkaisuaika pienentyi 15,9 % prosenttia verrattuna instanssin ($m = 20$, $W = 700$) ratkaisuaikaan nähden. Toisaalta tapauksessa ($m = 30$, $W = 900$) ratkaisuaika kasvoi 70 % tapaukseen ($m = 20$, $W = 900$) verrattuna. Parametreilla on siis laskenta-aikaa lyhentävä vaikutus, kun $m = 30$, mutta sen huomaaminen ei olisi ollut mahdollista tarkastelemalla pelkästään tapauksia ($m = 30$, $W = 900$) ja ($m = 20$, $W = 900$).

Jos tarkastellaan sarakkeita generoivan algoritmin laskenta-ajan kasvua, kun muuttujan m arvoa kasvatettiin 10:llä, huomataan, että kasvu oli poikkeuksellisen suurta verrattaessa instansseja ($m = 30$, $800 \leq W \leq 900$) ja ($m = 40$, $800 \leq W \leq 900$) toisiinsa. Tapauksen ($m = 40$, $W = 900$) laskenta-aika oli 1280 % suurempi kuin tapauksen ($m = 30$, $W = 900$) ja tapauksen ($m = 40$, $W = 800$) laskenta-aika 520 % suurempi kuin tapauksen ($m = 30$, $W = 800$). Parametrien vaikutus ei kuitenkaan näkynyt näissä tapauksissa yhtä selvästi kuin brute-force-menetelmän kohdalla.

6.3.3 Poikkeavat havainnot

Poikkeavana havaintona sarakkeita generoivan algoritmin kohdalla voidaan pitää instanssia ($m = 90$, $W = 900$). Tapauksissa ($m = 90$, $500 \leq W \leq 800$) laskenta-aika kasvoi yli 200 % verrattuna tapauksiin ($m = 80$, $500 \leq W \leq 800$), mutta instanssin ($m = 90$, $W = 900$) kohdalla laskenta-aika pieneni 3 % tapaukseen ($m = 80$, $W = 900$) nähden. Taulukon 2 tulosten tarkastelu osoittaa, että poikkeavassa tapauksessa käytettyjen leikkausmuottien lukumäärä ei juurikaan muuttunut tapaukseen ($m = 80$, $W = 900$) nähden, mikä selittää laskenta-aikojen pienen eron, mutta ei kerro sen syytä. Käytettyjen leikkausmuottien lukumäärä ei siis aina tunnu riippuvan yhtä voimakkaasti muuttujista m ja W.

7 Yhteenveto ja pohdinnat

Tarkasteltujen LPM-instanssien ratkaisuaika riippui käytetystä menetelmästä, tehtävän koosta sekä generoidun tehtävän parametreista. Brute-force-menetelmä ratkaisi nopeammin pienen kokoluokan tehtävät, kun taas sarakkeita generoiva algoritmi oli tehokkaampi isoissa tehtävissä. Sarakkeita generoiva algoritmi ratkaisi myös sellaiset tehtävät, joita brute-force ei kyennyt ratkaisemaan 5 tunnin aikarajaan mennessä.

Useimmissa tapauksissa rullien lukumäärän m ja emorullan koon W kasvattaminen lisäsi brute-force-menetelmän ja sarakkeita generoivan algoritmin käytettyjen leikkausmuottien lukumäärää sekä pidensi tehtävän ratkaisemiseen tarvittavaa laskenta-aikaa. Leikkausmuottien lukumäärän ja laskenta-ajan välillä havaittiin siis voimakas positiivinen riippuvuus.

Erityisesti muuttujan W kasvattaminen lisäsi laskenta-aikaa. Muuttujan m päävaikutus oli myös laskenta-aikaa pidentävä, vaikka päävaikutuksen voimakkuutta väärensi kyseisen muuttujan sekä parametrien b_i ja w_i, $i = 1, \ldots, m$, yhteisvaikutus. Laajempi koeasetelma sekä laskenta-aikojen mittaaminen useilla eri parametrien arvoilla olisi mahdollistanut tilastollisen testauksen ja siten luotettavamman analyysin brute-force-menetelmän ja sarakkeita generoivan algoritmin laskenta-ajoista. Tällöin myös pää- ja yhteisvaikutuksen ero olisi ollut paremmin havaittavissa. Työn pituuden puitteissa tämän laajemman koeasetelman tutkiminen ei kuitenkaan ollut mahdollista.

7.1 Parametrien valinta

Tarkastelluissa LPM-instansseissa muuttujiksi oli valittu rullien lukumäärä m ja emorullan koko W sekä parametreiksi rullakoko w_i ja kysyntä b_i, $i = 1, \ldots, m$. On tärkeää huomata, että muuttujiksi olisi voitu valita myös rullakoko ja kysyntä ja parametreina pidetty sen sijaan suureita m ja W. Suureiden m ja W valintaa muuttujiksi pidettiin kuitenkin luontevana, sillä sarakkeita generoivan algoritmin kompleksisuus kasvaa voimakkaasti näiden funktiona sekä simplex-algoritmin suorittamien laskuoperaatioiden määrä riippuu tekijästä m. Lisäksi m ja W ovat yksidimensioisia suureita, jolloin niiden käsittely ja tulosten esittäminen on helpompaa kuin m-dimensioisten rullakoko- ja kysyntäsuureiden.

Muuttujien arvojoukot $m \in \{10, 20, \ldots, 100\}$, ja $W \in \{500, 600, 700, 800, 900\}$ sekä parametrien w_i ja b_i vaihteluvälit $[l_w, u_w] = [100, 500]$, ja $[l_b, u_b] = [0, 100]$ valittiin useiden eri kokeilujen tuloksena. Ongelmaksi muodostui erityisesti lukujen l_w ja W_{max} määrittäminen, missä W_{max} on suurin W:n arvo. Jos suhde W_{max}/l_w oli liian suuri, ei brute-force-menetelmä kyennyt 5 tunnissa ratkaisemaan LPM-instansseja pienilläkään m:n arvoilla. Jos suhde oli taas liian pieni, oli tehtävien ratkaiseminen turhankin nopeaa brute-force-menetelmällä, jotta laskenta-aikoja oltaisiin voitu analysoida tulos-osiossa käsitellyllä tavalla.

Parametrien w_i, $i = 1, \ldots, m$, määrityksessä käytetty satunnaislukugeneraattori `randi` saattoi arpoa saman kokonaisluvun useammalla indeksin i arvolla, siten että $w_i = w_j$ jollakin $i \neq j$, missä $i, j \in \{1, \ldots, m\}$. Tällaisessa tapauksessa rajoitusmat-

riisin $\mathbf{A}$ rivivektorit $\mathbf{a}'_i$ ja $\mathbf{a}'_i$ ovat lineaarisesti riippuvia, jolloin tehtävän ratkaiseminen helpottui. Parametrin w_i vaihteluväli $[100, 500]$ oli kuitenkin asetettu sen verran suureksi, että kyseisen tilanteen sattuminen useasti ei ollut kovin todennäköistä.

7.2 Brute-force-menetelmä

Brute-force-menetelmän laskenta-aika LPM-instanssien ratkaisemisessa kasvoi erittäin voimakkaasti muuttujien m ja W funktiona, siten että muuttujan W vaikutus oli keskimäärin muuttujan m vaikutusta suurempi. Käytettyjen leikkausmuottien lukumäärän ja laskenta-ajan välillä vallitsi lisäksi vahva positiivinen korrelaatio. Parametrien b_i ja w_i vaikutus laskenta-aikaan pystyttiin selvästi tunnistamaan muutamassa tilanteessa.

Brute-force-menetelmä oli useimmissa LPM-instansseissa sarakkeita generoivaa algoritmia nopeampi menetelmä, mikä kertoo siitä, että tarkasteltavat instanssit olivat enimmäkseen pieniä ja siten brute-force-menetelmälle paremmin soveltuvia. Jos muuttujien m ja W arvojoukot sekä parametrien vaihteluvälit olisivat olleet suurempia, olisivat tuloksetkin olleet erilaisia.

On tärkeää huomata, että brute-force-menetelmälle asetettu 5 tunnin aikaraja teki keskeytettyjen instanssien vertailun mahdottomaksi, eli arvoitukseksi jäi kuinka paljon kyseisten instanssien ratkaisuajoissa olisi ollut eroa. Näissä tapauksissa myös brute-force-menetelmän ja sarakkeita generoivan algoritmin todellinen laskentaero jäi epäselväksi. Pidempi aikaraja olisi mahdollistanut menetelmien paremman tehokkuusvertailun, mutta samalla nostanut merkittävästi käytettyä kokonaislaskenta-aikaa. Yli 10 tunnin aikarajan asettaminen ei kuitenkaan olisi tuonut enää lisähyötyä lyhempiin aikarajoihin nähden, sillä kokeellisesti havaittiin MATLABin muistin loppuvan, jos instanssin ratkaiseminen oli kestänyt yli 10 tuntia. Ongelmallisuutta lisäsi lisäksi se, että ajon kestoa ei kyetty arvioimaan etukäteen, koska leikkausmuottien määrän ja tehtävän koon välistä riippuvuutta ei tunnettu.

7.3 Sarakkeita generoiva algoritmi

Sarakkeita generoivan algoritmin laskenta-aika LPM-instanssien ratkaisemisessa kasvoi voimakkaasti muuttujien m ja W funktiona. Muuttujan W vaikutuksen havaittiin olevan keskimäärin muuttujan m vaikutusta suurempi. Käytettyjen leikkausmuottien lukumäärän ja laskenta-ajan välillä vallitsi lisäksi vahva positiivinen korrelaatio. Parametrien b_i ja w_i aiheuttama vaikutus ei näkynyt tuloksissa yhtä selvästi kuin brute-force-menetelmän kohdalla.

Sarakkeita generoivan algoritmin rajoitusmatriisin $\tilde{\mathbf{A}}_k$ alkuarvauksena käytettiin yksikkömatriisia $\mathbf{I}_m$. Osa yksikkömatriisia vastaavista sarakkeista oli kuitenkin turhia, jos algoritmin edetessä löytyi sellaisia leikkausmuotteja, jotka sisälsivät enemmän kuin yhden pienen rullan. Tämä seuraa siitä, että leikkausmuotti $\mathbf{A}_j$ on turha, jos on olemassa toinen muotti $\mathbf{A}_l$ siten, että $a_{ij} < a_{il}$ jollakin $i \in \{1, \ldots, m\}$. Turhien sarakkeiden poistaminen tapahtui CPLEXin esikäsittelyssä, mutta se tehtiin joka RLPM-tehtävän ratkaisemisen yhteydessä, kun turha sarake olisi kerralla

voitu poistaa rajoitusmatriisista $\tilde{\mathbf{A}}_k$. Epäselväksi jäi siis, kuinka paljon algoritmin toiminta hidastui näiden ylimääräisten välivaiheiden takia.

Tarkastelluissa LPM-instansseissa sarakkeita generoivalla algoritmilla kului enintään 2h 16min tehtävän ratkaisemiseen. Mielenkiintoista olisi ollut nähdä, kuinka paljon muuttujien m ja W arvoja olisi pitänyt kasvattaa, jotta 5 tunnin aikaraja olisi tullut vastaan.

Viitteet

Barvinok, A. 2008. *Integer Points in Polyhedra*. An Arbor, Michigan, USA: European Mathematical Society. 200. ISBN 978-3-03719-052-4.

Bertsimas, D. & Tsitsiklis, J.N. 1997. *Introduction to Linear Optimization*. 3rd ed. Belmont, Massachusetts, USA: Athena Scientific. 608. (Athena Scientific optimization and computation series 7). ISBN 1-886529-19-1.

Chvátal, V. 1983. *Linear Programming*. 1st ed. New York, New York, USA: W.H. Freeman. 478. ISBN 978-0-7167-1587-0.

Dantzig, G.B. 1951. *Maximization of a linear function of variables subject to linear inequalities*. Teoksessa: Koopmans, T.C. (toim.) *Activity Analysis of Production and Allocation*, New York, New York, USA: John Wiley & Sons. S. 339-347.

Gilmore, P.C. & Gomory, R.E. 1961. *A linear programming approach to the cutting-stock problem*. Operations Research. Vol. 9:6. S. 849-859.

Gilmore, P.C. & Gomory, R.E. 1963. *A linear programming approach to the cutting stock problem - part II*. Operations Research. Vol. 11:6. S. 863-888.

Kantorovich, L. 1960. *Mathematical Methods of Organizing and Planning Production*. Management Science. Vol. 6:4. S. 366-422. (DOI: 10.1287/mnsc.6.4.366).

Land, A.H. & Doig, A.G. 1960. *An automatic method of solving discrete programming problems*. Econometrica. Vol. 28:3. S. 497-520. ISSN 00129682. (DOI:10.2307/1910129).

Shimizu, K, & Ishizuka, Y, & Bard, J.F. *Nondifferentiable and Two-Level Mathematical Programming*. Norwell, Massachusetts, USA: Kluwer Academic Publishers. 470. ISBN-10: 0792398211.

Woeginger, G.J. 2003. *Exact Algorithms for NP-Hard Problems: A Survey*. Combinatorial Optimization - Eureka, You Shrink! Lecture notes in computer science. Vol. 2570. S. 185-207.

Liite A

Taulukko 1: Sarakkeita generoivan algoritmin (SG) ja brute-force-menetelmän (BF) ratkaisuajat LPM-instansseilla. Jos instanssia ei ollut kyetty ratkaisemaan 5 tunnissa, keskeytettiin algoritmin toiminta ja tulokselle käytettiin merkintää #. Lisäksi sinisellä värillä on merkitty tapauksia, joissa brute-forcen laskenta-aika oli 45-100 % sarakkeita generoivan algoritmin laskenta-ajasta ja punaisella värillä tapauksia, joissa sarakkeita generoiva algoritmi oli nopeampi.

$m \backslash W$	500		600		700		800		900	
	BF	SG	BF	SG	BF	SG	BF	SG	BF	SG
10	0,14s	2,21s	0,13s	2,31s	0,12s	2,84s	0,13s	2,36s	0,14s	2,75s
20	0,14s	3,04s	0,17s	4,11s	0,21s	6,04s	0,30s	7,20s	0,55s	10,8s
30	0,15s	2,21s	0,20s	4,45s	0,27s	5,08s	0,46s	10,5s	1,19s	18,4s
40	0,35s	5,55s	0,90s	8,25s	5,83s	24,1s	39,8s	1min 5s	4min 25s	4min 16s
50	0,82s	6,68s	6,82s	13,9s	1min 6s	30,0s	9min 44s	2min 34s	1h 23min	6min 1s
60	1,67s	8,42s	9,76s	49,3s	1min 9s	2min 24s	10min 49s	10min 16s	1h 39min	24min 43s
70	2,36s	26,1s	25,1s	1min 57s	4min 47s	9min 26s	1h 22min	22min 29s	#	56min 20s
80	7,00s	28,1s	1min 25s	1min 36s	17min 41s	8min 4s	3h 57min	31min 33s	#	2h 16min
90	20,8s	1min 46s	5min 44s	9min 4s	1h 42min	25min 0s	#	1h 40min	#	2h 12min
100	7,37s	2min 40s	1min 41s	11min 13s	26min 4s	27min 28s	#	1h 42min	#	2h 9min

Liite B

Taulukko 2: Sarakkeita generoivan algoritmin (SG) ja brute-force-menetelmän (BF) käyttämien leikkausmuottien lukumäärä sekä niiden teoreettinen yläraja N LPM-instansseissa. Jos instanssia ei ollut kyetty ratkaisemaan 5 tunnissa, keskeytettiin algoritmin toiminta ja lukumäärälle käytettiin merkintää #.

$m\backslash W$	500			600			700			800			900		
	BF	SG	N	BF	SG	N	BF	SG	N	BF	SG	N	BF	SG	N
10	17	14	3072	25	16	3072	45	17	4608	80	22	23328	125	23	34992
20	135	45	$6*10^{7}$	317	47	$5*10^{8}$	728	58	$4*10^{9}$	1576	63	$1*10^{10}$	3279	73	$8*10^{10}$
30	105	44	$3*10^{10}$	275	60	$6*10^{11}$	656	70	$9*10^{12}$	1500	82	$2*10^{14}$	3335	96	$5*10^{15}$
40	736	78	$2*10^{16}$	2231	108	$4*10^{17}$	6877	120	$2*10^{19}$	18842	146	$5*10^{21}$	50862	172	$9*10^{22}$
50	1766	96	$2*10^{20}$	6577	134	$4*10^{22}$	22043	143	$2*10^{24}$	68409	187	$2*10^{26}$	203616	202	$1*10^{29}$
60	1547	130	$2*10^{23}$	5754	158	$9*10^{25}$	19887	202	$5*10^{28}$	64585	246	$5*10^{31}$	198924	280	$6*10^{33}$
70	2743	156	$8*10^{27}$	10761	203	$5*10^{30}$	39690	262	$2*10^{33}$	138821	290	$9*10^{36}$	#	317	$2*10^{40}$
80	4253	195	$2*10^{31}$	18184	204	$5*10^{34}$	72032	280	$1*10^{38}$	267700	330	$6*10^{41}$	#	373	$3*10^{45}$
90	8286	234	$4*10^{36}$	38336	298	$2*10^{40}$	166152	341	$7*10^{43}$	#	393	$1*10^{48}$	#	379	$9*10^{52}$
100	4267	244	$8*10^{37}$	19093	308	$1*10^{42}$	79477	403	$5*10^{46}$	#	351	$5*10^{50}$	#	389	$7*10^{54}$