CHALMERS

GÖTEBORGS UNIVERSITET

Matematisk optimering för leveransplanering av sockerbetor

Mathematical optimisation for transport scheduling of sugar beets

Kandidatarbete inom civilingenjörsutbildningen vid Chalmers

Viktor Göransson
Oscar Nilsson
Elias Torstensson

Institutionen för Matematiska vetenskaper
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET
Göteborg, Sverige 2023

Matematisk optimering för leveransplanering av sockerbetor

Kandidatarbete i matematik inom civilingenjörsprogrammet Teknisk matematik vid Chalmers

Viktor Göransson Oscar Nilsson Elias Torstensson

Handledare: Axel Ringh

Institutionen för Matematiska vetenskaper
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET
Göteborg, Sverige 2023

Förord

Denna kandidatuppsats är skriven vid Institutionen för Matematiska vetenskaper på Chalmers tekniska högskola. Vi vill främst tacka vår handledare Axel Ringh som väglett oss genom processen och givit oss värdefulla insikter under arbetets gång. Vi vill också tacka Nordic Sugar som givit oss deras stöd och tillåtelse att använda deras data. Vi tackar även Betfrakt som bidragit med insikt i deras arbetsprocess och givit oss kontinuerlig feedback.

Nedan följer en redovisning av vem som är huvudförfattare till vilken del av rapporten och till vilka delar man gjort ett bidrag till. Det är däremot viktigt att poängtera att samtliga i gruppen har arbetat tillsammans, arbetet har varit iterativt och krävt att alla bidragit med sina förmågor. Utöver nedanstående redovisning finns tidsloggar som indikerar vad som gjorts, och när. Varje vecka har även en dagbok förts, ansvaret för denna har roterat varje vecka.

- **Viktor Göransson:** Jag har huvudförfattat *Populärvetenskaplig presentation* och delförfattat *Sammandrag*, specifikt översättningen av sammandraget till engelska och revidering. Jag har också delförfattat 1.1 med beskrivningar av lastarens egenskaper och villkor som gäller vid lastning. Jag är huvudförfattare till 3.1, 3.2, 3.3. I dessa kapitel har jag introducerat mängderna och parametrar samt binära variabler f, z och u. Till dessa variabler har jag även formulerat bivillkoren. Dessa kapitel har jag även reviderat. Jag är delförfattare till kapitel 3.4, specifikt införandet av w-variablerna och dess bivillkor, härledningen av parametern v och en del av införandet av s-variablerna och dess bivillkor. I kapitel 3.5 och 3.6 har jag främst reviderat och strukturerat ordningen på texten. I 4.1 har jag varit huvudförfattare. I kapitlet har jag formulerat den introducerande texten. Vidare har jag tillsammans med Oscar formulerat scenarion i 4.1.2 och 4.1.4. Jag har även formulerat scenarierna 4.1.1 och 4.1.3. Till samtliga scenarion i detta kapitel har jag formulerat bakgrundsinformationen och delar av motiveringarna. Jag har även skapat figurerna i kapitlet och reviderat kapitlet. Jag har delförfattat 4.2, specifikt härledningen av ordningstalet i det mindre verkliga exemplet. Jag är huvudförfattare till 5 och har skrivit majoriteten av texten och reviderat denna. Jag är delförfattare till 6 och har reviderat samt skrivit på motiveringar av α och β tillsammans med huvudförfattaren. Jag har skrivit delar av koden till indelningen av fält i B.1. Jag har också skrivit koden för beräkningen av transporttid mellan fält i B.2. Till sist har jag skrivit koden till implementeringen av de fyra scenarierna i 4.1 och det verkliga problemet, se B.5 samt B.6.
- **Oscar Nilsson:** Jag har varit delförfattare till *Populärvetenskaplig presentation* där jag bidragit till diskussionen kring för- och nackdelarna med matematisk optimering kontra manuell lösning av transportproblem. Jag har även reviderat avsnittet. I *Sammandrag* har jag varit huvudförfattare. Där har jag sammanfattat arbetet och i samråd med de andra tagit fram lämpliga nyckelord. I kapitel 1 har jag varit huvudförfattare till 1.1 och 1.2. Jag har dels skrivit stora delar av texten som beskriver problemet, men även varit ansvarig för att revidera innehållet. Jag även delförfattare till 1.3 där jag varit med och formulerat samt reviderat avgränsningarna. För hela kapitel 2 är det jag som är huvudförfattare. Jag har formulerat den matematiska teorin och skrivit stor del av modellexemplet. Det är även jag som har skrivit Matlab-koden för modellexemplet i B.4. I den matematiska modellen har jag varit delförfattare till 3.1 och 3.2 samt varit huvudförfattare till kapitel 3.5 och 3.7. Det är även jag som har skrivit till Matlab-koden som beräknar big-M villkoren och ordningstalet H i B.3, jag härleder ordningstalet H i 4.2. I 3.1 och 3.2 har jag varit med och formulerat texten. I 3.5 har jag varit ansvarig för att formulera och motivera bivillkoren som rör den dagliga kvoten. I den sammanfattade modellen i 3.7 har jag varit ansvarig för att sammanfatta innehållet i kapitel 3 och formulerat det enligt den notation vi introducerat i kapitel 1. Det är jag som är huvudförfattare till inledningen i kapitel 4. Där har jag introducerat implementationen och hur problemet löses numeriskt. Jag har även varit med och formulerat scenarion i 4.1.2 och 4.1.4. Inom kapitel 5 har jag varit med och formulerat samt reviderat de etiska och miljömässiga aspekterna. I diskussionen i kapitel 6 har jag främst varit med och utvärderat och reviderat innehållet. Det är jag som stått för referenshanteringen. Slutligen är det jag som har strukturerat och formatterat Appendix A och B.

- **Elias Torstensson:** Jag är huvudförfattare till avsnitt 1.3, där jag beskrev de avgränsningarna som gjorts. I avsnitt 3.4 skrev jag majoriteten, mer precist införandet av κ-variablerna och motiveringen av dess bivillkor. Jag har även skrivit delar av motiveringen av s-variablerna och dess bivillkor. I avsnitt 3.6 har jag skrivit stora delar, bland annat motiveringarna av termerna i målfunktionen och resonemang kring straffkonstanter. Mycket av 4.2 har jag skrivit. Bland annat diskussionerna kring valet av ordningstal. Diskussionen är huvudsakligen skriven av mig, i sammarbete med Viktor. Jag har skrivit hela Python-implementeringen av modellen, och stora delar av koden till fältindeldningen. Har även huvudförfattat texten i appendix om hur fältindelningen sker. I avsnitt 2.2 har jag skrivit en bit av introduktionen till optimering och exempelproblemet. Vissa delar av bakgrunden till problemet har jag också skrivit. I avsnitt 3.3 skrev jag delen om x-variablerna och tillhörande bivillkor. I avsnitt 4.1 skrev jag delar av motiveringarna till varför lösningen är optimal.

Populärvetenskaplig presentation

Människan försöker ofta hitta den mest effektiva vägen genom vardagen. Vilken väg vi går genom mataffären eller hur en budbilsförare levererar paket är exempel på detta. Om en budbilsförare har fem paket att leverera under en dag kan man fundera på vilken leveransordning som är mest fördelaktig. Budbilsföraren vill göra ett *optimalt* leveransschema. En viktig fråga är däremot vad föraren vill göra schemat optimalt med avseende på. Det kan exempelvis vara kortast körsträcka eller snabbast rutt. Den kortaste körsträckan kan vara längs landsvägar, men den snabbaste via motorvägar. Optimala scheman kan därav se olika ut beroende på vilka mål man har. Det kan även finnas krav på hur saker ska utföras. Exempelvis arbetar föraren mellan bestämda tider på dygnet, vilket gör att paket inte kan levereras utanför dessa tider. Ett sådant krav gör att antalet möjliga scheman begränsas till ett visst antal kombinatoriska möjligheter, av vilka man vill hitta det optimala schemat. Däremot, om man ställer upp för många eller för stränga krav, finns det eventuellt inte något schema som uppfyller kraven. I det fallet kan vi inte hitta ett optimalt schema.

Ett schema kan till exempel göras utifrån erfarenhet, eller med hjälp av matematisk optimering. Det finns för- och nackdelar i båda fallen. Ett schema som skapas manuellt utifrån erfarenhet inom området kan ta större hänsyn till vad som fungerar i verkligheten. Däremot kan komplexiteten öka markant om man skalar upp problemet. Det kan i så fall innebära att problemet inte går att lösa för hand. Om budbilsföraren i exemplet ovan i stället ska planera leveranser av 100 paket under en månad behöver svaret inte längre vara uppenbart. En människa har också svårt att visualisera lösningar till stora och komplexa problem. Fördelen med matematisk optimering är dess förmåga att översätta verkligheten till matematik. Med hjälp av datorverktyg kan eventuellt en lösning hittas till problem där den mänskliga beräkningskraften inte räcker till. Den huvudsakliga nackdelen är däremot att det är svårt att översätta verkligheten till matematik på ett realistiskt och representativt vis.

Arbetet härleder en matematisk modell som beskriver i vilken ordning en lastare transporteras mellan fält för att lasta sockerbetor till lastbilar som transporterar dem till ett sockerbruk. Den matematiska modellen implementeras sedan i en lösare för att hitta ett optimalt schema för lastaren med avseende på kostnaden. I analogin med budbilen kan man se det som att paketen (sockerbetorna) i stället ska hämtas på olika platser (fält) och lämnas på en gemensam plats (sockerbruket).

Sammandrag

Kandidatarbetet tar fram en matematisk modell som beskriver schemat för en lastare, vars uppdrag är att lasta sockerbetor till lastbilar som transporterar dem till ett sockerbruk. Modellen formuleras som ett linjärt blandat heltalsprogram från en beskrivning av verksamheten. Numerisk lösning av modellen sker via en implementation i lösaren CPLEX. Modellen verifieras genom att skapa fyra stycken delproblem som löses analytiskt och numeriskt där svaren jämförs. Avslutningsvis diskuteras modellens lösning och tidskomplexitet inklusive lösningstid sett till de modelleringsval och parameterval som gjorts.

***Nyckelord*:** heltalsprogrammering, linjärprogrammering, matematisk modellering, optimering, schemaläggning

Abstract

The bachelor's thesis derives a mathematical model that describes how a loader is scheduled to load sugar beets onto trucks that transports them to a sugar factory. The model is formulated as a mixed integer linear program according to a description of the operation. The model is implemented in CPLEX and solved numerically. Four subproblems are created and solved analytically and numerically to verify the model. The solutions are then compared. Lastly, the solution and time complexity of the model is discussed with respect to modelling and parameter choices.

***Key words*:** integer programming, linear programming, mathematical modelling, optimization, scheduling

Innehåll

1 Inledning

1.1 Bakgrund till problemet

Nordic Sugar är ett företag som driver ett sockerbruk i Skåne. Under hösten och vintern tar sockerbruket in en viss mängd sockerbetor varje dag från fält runt om i Skåne. En lastare används för att lasta sockerbetor till lastbilar som transporterar dem till sockerbruket. Denna lastare kör från fält till fält, tills alla sockerbetor på alla fält är lastade och transporterade till sockerbruket. Majoriteten av de rörliga transportkostnaderna beror på lastaren. Således planerar man i vilken ordning lastaren kör mellan fälten för att få en så låg rörlig kostnad för lastaren som möjligt. Denna planering görs idag för hand. I detta arbete tar vi fram en matematisk modell för hur lastaren ska förflyttas för att få en så låg rörlig kostnad som möjligt.

För att underlätta samordning av transporter delas Skåne in i administrativa enheter, hädanefter benämnda som *leveransgrupper*. Varje leveransgrupp har ett företag kopplat till sig som agerar mellanhand mellan odlare och Nordic Sugar. Schemaläggning av sockerbetsleveranser från fält till sockerbruk utförs av respektive företag i varje leveransgrupp.

Det finns en lastare knuten till varje sammanhängande territorium av fält, vidare benämns dessa territorium som *områden*. Vilka fält som ingår i ett visst område bestäms av Nordic Sugar i samråd med leveransgrupperna. I vårt arbete fokuserar vi på område Sydväst, som tillhör leveransgrupp Nordvästra Skåne, se figur 6 i A.4. Företaget som ansvarar för denna leveransgrupp är Betfrakt.

En *kampanj* pågår från september till januari, under denna tid tas sockerbetor upp från fälten och transporteras till sockerbruket. Kampanjen är uppdelad i fem stycken *leveransperioder*. Varje leveransperiod består av cirka 30 dagar, där ett visst antal ton sockerbetor ska levereras till sockerbruket från varje område. För varje område i respektive leveransgrupp delas fälten in i de fem olika leveransperioderna. En mer detaljerad diskussion kring indelningen förs i A.4.

På varje fält finns en *stuka*, vilket är en samling sockerbetor. En lastare fyller lastbilar med sockerbetor från en stuka med en lastningsfart av 163 ton/h och kör mellan fälten i 20 km/h. Driftkostnaden för lastaren är 1300 kr/h. Denna kostnad inkluderar bränsle, underhåll av lastaren och maskinförarens lön.

En arbetsdag är tio timmar lång, varje arbetsdag behöver sockerbruket en viss mängd[1] sockerbetor. Om ett fält är så stort att det inte hinner lastas klart under en dag stannar lastaren kvar på fältet under natten och fortsätter nästkommande dag. Ifall ett fält har lastats klart men lastaren inte hinner flytta till nästa fält under arbetsdagen, transporteras lastaren till nästa fält efter ordinarie arbetstid. Det sker för att lastaren ska vara på plats och börja lasta direkt på morgonen dagen efter. Att transportera lastaren efter arbetsdagens slut medför inga extra kostnader utöver den ordinarie transportkostnaden. Om den dagliga kvoten till sockerbruket är uppfylld innan arbetsdagen är slut, går maskinförarna hem tidigare. Det är tillåtet att lämna en liten mängd sockerbetor på ett fält, detta diskuteras i avsnitt 3.4.

Målet med detta kandidatarbete är att formulera en matematisk modell som beskriver hur lastaren transporteras mellan odlares fält under en kampanj. Den framtagna modellen implementeras och löses numeriskt för att hitta ett optimalt transportschema som beskriver i vilken ordning lastaren ska lasta stukorna för att minimera den rörliga kostnaden för lastaren under en leveransperiod. Idag utförs denna leveransplanering manuellt utifrån erfarenhet från tidigare års planeringar. Det är därför intressant för Betfrakt och Nordic Sugar att utforska möjligheten till leveransplanering med hjälp av optimeringsverktyg för att eventuellt minska de rörliga kostnaderna i framtiden.

1.2 Problemställning

Det huvudsakliga problemet är att ta fram och verifiera en matematisk modell som beskriver de rörliga kostnaderna under varje leveransperiod för lastaren i ett område. Modellen formuleras som ett linjärt blandat heltalsprogram.

[1] mängden refererar till antalet ton

1.3 Avgränsningar

Verkliga problem är ofta väldigt komplexa. Eftersom huvuddelen av arbetet består av att ställa upp en matematisk modell av ett verkligt problem behöver avgränsningar införas.

Vi beaktar ej kostnaden för lastbilstransporter mellan fälten och fabriken. Lastbilarna kör mellan fält och sockerbruket, den totala sträckan som lastbilarna kör antas därmed vara konstant oavsett lastarens körväg. Scheman för de individuella lastbilarna beaktas ej, det antas skötas externt givet ett visst schema för lastaren.

Den mängd sockerbetor som sockerbruket önskar få levererat till sig varje dag kan i verkligheten variera under dagen på grund av störningar i driften. I vår modell antas kvoten vara konstant under en hel dag. Det är med det antagandet schemat görs av Betfrakt.

Arbetet behandlar endast en leveransgrupp och ett område. I övriga områden transporterar lastbilarna mer än bara sockerbetor. Att beakta detta skulle leda till en betydligt mer komplex modell, och ligger inte inom ramen för detta arbete.

På grund av begränsad datorkraft har vi inte möjlighet att beräkna ett optimalt schema för en hel kampanj. Därför ställer vi upp och löser ett problem som använder verklig data, men som inte har lika många fält och dagar som en hel kampanj.

2 Matematisk optimering

Detta avsnitt ger en introduktion till matematisk optimering för den läsare som är ny i ämnet. Vi täcker grundläggande teori och presenterar hur ett enkelt modellproblem löses i praktiken. Teorin är tillräcklig för att kunna följa resten av rapporten utan svårigheter, men mer avancerad teori utelämnas.

2.1 Grundläggande begrepp

Den centrala idén inom matematisk optimering är att, givet ett problem, i någon mån hitta bästa möjliga lösning. För att hitta denna lösning väljs systematiskt värden på *variabler* från en mängd. Det finns ofta krav på hur variabler ska förhålla sig till varandra, men även till specifika värden. Dessa krav representeras av så kallade *bivillkor*. Värdena på variablerna påverkar i sin tur en funktion som mäter hur väl de valts. Denna funktion kallas för *målfunktion* och är således det man vill minimera, eller maximera. Följande avsnitt är baserat på [1].

Låt $f : \mathbb{R}^n \to \mathbb{R}$ beteckna en generell målfunktion, $g_i : \mathbb{R}^n \to \mathbb{R},\ \forall i \in \mathcal{I}$ generella olikhetsbivillkor, och $h_j : \mathbb{R}^n \to \mathbb{R},\ \forall i \in \mathcal{E}$ generella likhetsbivillkor, där $\mathcal{I}$ och $\mathcal{E}$ är mängderna av alla olikhets- respektive likhetsbivillkor. Minimeringsproblemet skrivs då som

$$\begin{aligned} \min \quad & f(\mathbf{x}), \\ \text{sådant att} \quad & g_i(\mathbf{x}) \leq 0,\ \forall i \in \mathcal{I} \\ & h_j(\mathbf{x}) = 0,\ \forall j \in \mathcal{E} \\ & \mathbf{x} \in \mathbb{R}^n. \end{aligned} \tag{1}$$

Det är denna notation för optimeringsproblemet som arbetet följer. Om man i stället vill formulera ett maximeringsproblem transformeras problemet genom en multiplikation av -1 i målfunktionen, och vice versa.

För att kunna fastställa att den lösning som erhålls är optimal i något avseende finns en rad villkor som behöver uppfyllas. Först och främst måste lösningen som erhålls vara tillåten. Låt $\mathcal{S} = \{\mathbf{x} \in \mathbb{R}^n : g_i(\mathbf{x}) \leq 0, h_j(\mathbf{x}) = 0,\ \forall i \in \mathcal{I},\ \forall j \in \mathcal{E}\}$ beteckna mängden av alla tillåtna lösningar i det generella fallet. Således måste en tillåten optimal lösning till (1) tillhöra mängden av de tillåtna lösningarna. För att precisera vad en optimal lösning innebär introduceras följande begrepp.

Definition 2.1 (Lokalt optimum) *En punkt $\boldsymbol{x}^* \in \mathcal{S}$ kallas för ett lokalt optimum till* (1) *om $f(\boldsymbol{x}^*) \leq f(\boldsymbol{x}),\ \forall \boldsymbol{x} \in S$ i en omgivning till $\boldsymbol{x}^*$.*

När man talar om en optimal lösning till ett problem är det ofta önskvärt att den är optimal i hela S. Hurvida en sådan lösning går att hitta beror på hur problemet ser ut. Det man letar efter är ett så kallat *globalt optimum*.

Definition 2.2 (Globalt optimum) *En punkt* $\boldsymbol{x}^*$ *kallas för ett globalt optimum till* (1) *om* $f(\boldsymbol{x}^*) \leq f(\boldsymbol{x}),\ \forall \boldsymbol{x} \in \mathcal{S}$.

Ett vanligt sätt att garantera att ett globalt optimum finns är via konvexitet. En konvex funktion karaktäriseras av att alla linjer mellan två distinkta punkter på funktionens graf ligger ovanför grafen. Om en konvex funktion $p(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}$ är två gånger kontinuerligt differentierbar, det vill säga $p(\mathbf{x}) \in \mathcal{C}^2(S),\ \forall \mathbf{x} \in S$, innebär det att Hessianen till funktionen är positivt semi-definit $\forall \mathbf{x} \in S$. Detta kan även appliceras på mängden $\mathcal{S}$. Om alla bivillkor $g_i(\mathbf{x})$ och $h_j(\mathbf{x})$ i (1) är positivt semi-definita $\forall \mathbf{x} \in S$ är den tillåtna mängden konvex. Ett konvext optimeringsproblem innebär att minimera värdet av en konvex målfunktion över en konvex mängd. Däremot, om problemet inte är konvext är ett lokalt optimalt värde ofta det bästa vi kan åstadkomma.

2.1.1 Linjär- och heltalsprogrammering

Det som idag på svenska kallas för *linjärprogrammering* presenterades under mitten av 1900-talet av George Dantzig i artikeln *Programming in a Linear Structure*. Dantzig forskade under den tiden på optimering inom schemaläggning hos det amerikanska flygvapnet. Termen programmering refererar till planer, eller föreslagna schemaläggingar inom till exempel logistik [3]. Det refererar alltså inte till dagens programmering som syftar på datorkod. Programmering lever kvar än idag som synonym till optimering. I arbetet använder vi programmering och optimering synonymt.

Det finns olika typer av matematisk programmering. De problem vi fokuserar på i denna rapport är linjära blandade heltalsprogrammeringsproblem. Vi börjar med att introducera två typer av matematisk programmering som tillsammans bidrar till de önskade egenskaperna hos linjär blandad heltalsprogrammering. Linjärprogrammering kan anses vara den enklaste typen av matematisk programmering. Dessa programmeringsproblem har en linjär målfunktion och samtliga bivillkor är linjära. Variablerna är kontinuerliga. Fördelen med denna typ av optimeringsproblem är dess relativt enkla geometri. För ett linjärt programmeringsproblem kommer en optimal lösning dessutom alltid vara globalt optimal på grund av konvexitet. *Heltalsprogrammering* är den typ av optimeringsproblem med egenskapen att alla, eller en del av variablerna endast antar heltalsvärden. Inom heltalsprogrammering behöver varken målfunktion eller bivillkor vara linjära. Med koncepten från linjärprogrammering och heltalsprogrammering introduceras *linjär blandad heltalsprogrammering*. Från linjärprogrammering har vi en linjär målfunktion såväl som linjära bivillkor. Från heltalsprogrammering innebär det att alla variabler, eller en del av variablerna endast kan anta heltalsvärden. Resterande variabler är kontinuerliga som för linjärprogrammering.

Vi introducerar nu en metod som används inom linjärprogrammering för att utvidga typen av problem som kan lösas. Denna metod kallas för big-M bivillkor och används dels för att diktera logiska uttryck [2], dels för att transformera olinjära bivillkor till linjära bivillkor. Det kan även användas för att transformera likhetsbivillkor till olikhetsbivillkor. Om big-M bivillkoret ska agera som en strömbrytare för binära variabler byggs bivillkoren upp av en koefficient M i ett bivillkor som väljs så att den är större än det högsta tänkbara värdet på uttrycket. Antag exempelvis att $x \in \mathbb{R}_+$ och $y \in \{0, 1\}$. Betrakta sedan bivillkoret $x \leq My$. Om $y = 0$ måste $x = 0$. Men om $y = 1$ ger det att $x \leq M$. Om M väljs stort nog att x begränsas av andra bivillkor medför detta att big-M villkoret inte inför nya begränsningar. Vi visar resterande egenskaper genom att transformera ett olinjärt likhetsbivillkor till ett linjärt olikhetsbivillkor. Antag att $x \in \mathbb{R}_+$ och $y \in \{0, 1\}$ som i föregående exempel. Betrakta nu det olinjära likhetsbivillkoret $xy = 0$. Om $y = 0$ medför det att $x \geq 0$. Om $y = 1$ medför det att $x = 0$. Ersätt bivillkoret med $x \leq M(1 - y)$. Då $y = 0$ medför det att $x \leq M$, om M återigen väljs stort nog att x i stället begränsas av andra bivillkor medför detta inte någon begränsning av x. Om $y = 1$ ger det att $x \leq 0$, det vill säga, $x = 0$. På så vis har vi nu ett linjärt olikhetsbivillkor. Vi använder dessa typer av big-M bivillkor i den matematiska modellen.

2.2 Ett exempelproblem

Vi introducerar koncepten från kapitel 2.1 genom ett enkelt illustrativt modelleringsproblem. Antag att en lantbrukare med ett 100 hektar stort fält vill planera sitt kommande år. Lantbrukaren har bestämt sig för att odla majs och vete. Optimeringsproblemet är att hitta den optimala mängden majs och vete som lantbrukaren ska odla för att maximera den totala vinsten.

Den beräknade vinsten per hektar är 24000 kronor för majs, och 4000 kronor för vete. Om lantbrukaren endast odlar en typ av gröda kan marken uttömmas på näring. För att se till att båda grödor odlas vill lantbrukaren att antalet hektar majs som odlas ska vara mindre eller lika med antalet hektar vete. Det finns även ett stort lager med frön kvar sedan förra året som hen önskar odla. Lantbrukaren vill således att minst 20 hektar av varje gröda ska odlas.

Exemplet har en optimal lösning som innebär att lantbrukaren odlar 50 hektar av varje gröda. Eftersom majs ger en större vinst per hektar är det mest lönsamt att odla så mycket majs som möjligt. På grund av risk för näringsuttömning måste lantbrukaren odla minst lika mycket vete som majs. För det 100 hektar stora fältet blir således den optimala lösningen att odla 50 hektar av varje gröda.

2.2.1 Modelluppställning av exempelproblemet

Vi demonstrerar nu hur exempelproblemet kan konverteras från text till en uppsättning av ekvationer som följer formatet i (1). Vi börjar med att definiera variablerna $x_v, x_m \in \mathbb{R}_+$ som representerar antalet hektar av varje frötyp som odlas. Från texten har vi en målfunktion $w(x_v, x_m) := 4000x_v + 24000x_m$, som representerar den totala vinsten. För att undvika att uttömma marken på näring har vi bivillkoret $x_m \leq x_v$. Eftersom lantbrukaren vill använda sitt frölager måste $x_v \geq 20$ och $x_m \geq 20$ då hen är tvungna att odla minst 20 hektar av varje gröda. Slutligen tar vi hänsyn till att fältet är 100 hektar stort, detta görs med bivillkoret $x_v + x_m \leq 100$. Notera att samtliga bivillkor sedan kan skrivas om på formen $g_i(x_v, x_m) \leq 0, \ \forall i \in \mathcal{I}$.

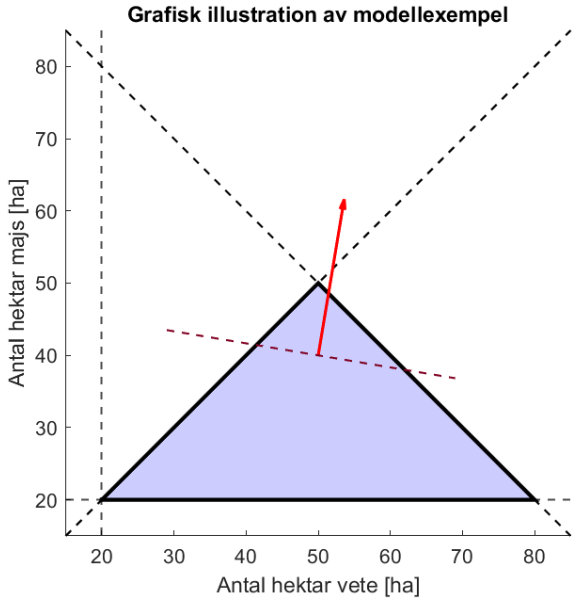
Således är maximeringsproblemet att

$$\begin{aligned} \max \quad & w(x_v, x_m) = 4000x_v + 24000x_m, \\ \text{sådant att} \quad & -x_v + x_m \leq 0, \\ & x_v + x_m - 100 \leq 0, \\ & -x_v + 20 \leq 0, \\ & -x_m + 20 \leq 0, \\ & x_v, \ x_m \in \mathbb{R}_+. \end{aligned} \tag{2}$$

Vi löser problemet geometriskt och verifierar att det är den globalt optimala lösningen. Då målfunktionen till problemet är linjär är $\nabla w(x_v, x_m)$ konstant. Längs sina nivåkurvor har målfunktionen ett konstant värde, dessa är ortogonala mot $\nabla w(x_v, x_m)$. Nivåkurvorna delar var och en in $x_v x_m$-planet i två slutna halvplan. Ett halvplan π_1 med punkter som ligger inom $\pm 90°$ av $\nabla w(x_v, x_m)$:s riktning, ett halvplan π_2 med punkter inom $\pm 90°$ av $-\nabla w(x_v, x_m)$:s riktning. Det är endast punkterna inom π_1 vi är intresserade av eftersom det endast är där målfunktionens värde ökar. Problemet är linjärt och tvådimensionellt, den tillåtna mängden S är dessutom begränsad. Därav utgör samtliga bivillkor tillsammans en polygon. Geometriskt finner vi problemets optimala lösning längs den nivåkurva som skär ett hörn i polygonen,[2] och som vid en förflyttning $\epsilon > 0$ i riktningen $\nabla w(x_v, x_m)$ inte har några punkter $(x_v, x_m) \in S$ kvar i π_1.

Exempelproblemet illustreras i figur 1. Där visualiseras exempelproblemets tillåtna mängd S tillsammans med en gradient $\nabla w(x_v, x_m)$ och en nivåkurva till $w(x_v, x_m)$. Lösningen vi erhåller motsvarar att 50 hektar av båda grödor odlas. Det är samma lösning vi resonerar oss fram till i 2.2. För denna lösning är målfunktionens värde 1.4 miljoner kronor. Målfunktionen och samtliga bivillkor är konvexa $\forall x_v, x_m \in S$. Eftersom problemet är konvext vet vi att det är den globalt optimala lösningen. Koden till exempelproblemet finns i Appendix B.4.

[2] nivåkurvan kan vara parallell med en kant beroende på problemets geometri, den optimala lösningen är då ej entydig.



Figur 1: Grafisk illustration av exempelproblemet i (2). Bivillkoren är markerade med sträckade svarta linjer. Det skuggade området inneslutet av de tjocka svarta linjerna är det tillåtna området $\mathcal{S}$. Den röda pilen pekar i riktningen av vektorfältet från $\nabla w(x_v, x_m)$. Den röda streckade linjen representerar en nivåkurva till $w(x_v, x_m)$.

3 Den matematiska modellen

I detta avsnitt härleder vi den matematiska modellen som beskriver de rörliga kostnaderna under varje leveransperiod för lastaren. Detta transportproblem ställs upp som ett optimeringsproblem, mer specifikt ett linjärt blandat heltalsprogrammeringsproblem. Målfunktionen som minimeras är den totala kostnaden för transporten av lastaren mellan fälten. Modellen skapas från informationen i avsnitt 1.1 och kan tillämpas på både en leveransperiod och en hel kampanj. För att formulera modellen likt systemet i (1) definieras först relevanta mängder. Sedan införs parametrar som behövs för att beskriva bivillkoren i modellen. Därefter införs binära variabler och deras bivillkor som behövs för att beskriva ett schema. För arbetstid, lastning och daglig kvot av betor till sockerbruket införs ytterligare variabler och bivillkor. Till sist formuleras målfunktionen och en sammanfattning av modellen ges.

3.1 Mängder

För att beskriva modellen införs först följande tre mängder:

- $I = \{1, 2, 3, \ldots, N\}$, mängden av fält som ska skördas,
- $J = \{1, 2, 3, \ldots, D\}$, mängden av dagar under en leveransperiod,
- $K = \{1, 2, 3, \ldots, H\}$, mängden av ordningstal.

Generellt använder vi gemener för att beteckna ett element i motsvarande mängd. Fältnummer $i \in I$ identifierar ett fält. En dag $j \in J$ beskriver en dag på kampanjen, det vill säga $j = 1$ betyder dag 1 på kampanjen. Ordningstalet $k \in K$ är en indikator på vilken ordning på en viss dag som en händelse sker, vilket preciseras i (3).

3.2 Parametrar

Härnäst definierar vi de parametrar som är nödvändiga i modellen. Relevanta parametrar för lastaren är:

- $c = 1300$ [SEK/h], kostnaden att transportera lastaren per timme,

- $h = 20$ [km/h], lastarens transporthastighet,
- $l = 163$ [ton/h], farten lastaren överför sockerbetor från stukan till lastbilen.

Härnäst definieras parametrar som berör individuella fält och kvoter:

- $q_j \in [0, \infty),\ \forall j \in J$ [ton], den mängd sockerbetor sockerbruket behöver få in dag j,
- $\psi_i \in [0, \infty),\ \forall i \in I$ [ton], mängden betor på fält i,
- $\tau_i \in [0, \infty]),\ \forall i \in I$ [h], tiden det tar att lasta stukan på fält i.

Till sist definieras parametrar som berör förhållanden mellan fält, avståndet och tiden det tar att köra med lastaren mellan två fält:

- $s_{i,p} \in [0, \infty),\ \forall i, p \in I$ [km], sträckan mellan fält i och fält p,
- $t_{i,p} \in [0, \infty]),\ \forall i, p \in I$ [h], tiden det tar att köra mellan fält i och fält p.

Tiden det tar att köra mellan fält kan beräknas som kvoten av avståndet mellan fälten och lastarens transporthastighet. Tiden det tar att lasta ett givet fält är kvoten av skörden för fältet och lastarfarten av sockerbetor. Dessa två samband kan skrivas som:

$$t_{i,p} = \frac{s_{i,p}}{h},\ \forall i, p \in I, \quad \text{respektive} \quad \tau_i = \frac{\psi_i}{l},\ \forall i \in I.$$

3.3 Binära variabler och deras bivillkor

För att beskriva ett schema matematiskt behövs en indikator på när stukan på ett fält ska lastas. Närmare bestämt kan ett schema beskrivas av en samling binära variabler som definierar på vilken dag, och givet en dag i vilken ordning den dagen, en stuka på ett visst fält ska lastas. Vi definierar dessa variabler som

$$f_{i,j,k} = \begin{cases} 1, & \text{om fält } i \text{ lastas på dag } j \text{ i ordning } k,\ \forall i \in I,\ \forall j \in J,\ \forall k \in K, \\ 0, & \text{annars.} \end{cases} \tag{3}$$

Indexet $k \in K$ används för att beskriva i vilken ordning lastaren kör mellan fält. Exempelvis representerar $f_{1,2,3}$ att fält ett på dag två är det tredje fältet för dagen som lastas. För att f-variablerna ska få önskade egenskaper introduceras ett antal bivillkor. Till exempel kan maximalt ett fält $i \in I$ lastas i ordning $k \in K$ under en dag $j \in J$. Matematiskt betyder det att

$$\sum_{i=1}^{N} f_{i,j,k} \leq 1,\ \forall j \in J,\ \forall k \in K. \tag{4}$$

Detta är ett olikhetsbivillkor, då det eventuellt inte finns några fält som ska lastas i ordning $k \in K$ på en viss dag. Vidare lastas ett fält $i \in I$ maximalt en gång på en godtycklig dag $j \in J$, därav är

$$\sum_{k=1}^{H} f_{i,j,k} \leq 1,\ \forall i \in I,\ \forall j \in J. \tag{5}$$

Det innebär att ett fält maximalt kan ha ett ordningstal under en dag, och lastaren kan därmed inte vara på samma fält mer än en gång per dag. Bivillkoret är kopplat till att lastaren lastar klart ett fält i taget. Som i föregående bivillkoret införs en olikhet, då ett specifikt fält inte nödvändigtvis lastas en specifik dag. För att säkerställa att fälten lastas i utsatt ordning införs följande bivillkor

$$\sum_{p=1}^{N} f_{p,j,k+1} \leq \sum_{i=1}^{N} f_{i,j,k},\ \forall j \in J,\ \forall k \in K \setminus \{H\}. \tag{6}$$

Bivillkoret ser till att ett fält inte kan lastas i till exempel ordning tre utan att ett annat fält lastats i ordning två under samma dag.

Modellen behöver ta hänsyn till vilka fält lastaren transporteras mellan. Om detta görs med hjälp av $f_{i,j,k}$ blir bivillkor i modellen olinjära. Därför införs en typ av variabler, som tar hänsyn till om lastaren transporteras mellan två fält på en viss dag i en viss ordning. Med dessa variabler blir modellen och bivillkoren linjära. De binära variablerna definieras som

$$x_{i,p,j,k} = \begin{cases} 1, & \text{om fält } i \text{ och } p \text{ körs i ordning } k \text{ respektive } k+1 \text{ på dag } j, \\ & \forall i, p \in I,\ \forall j \in J,\ \forall k \in K \setminus \{H\}, \\ 0, & \text{annars.} \end{cases} \tag{7}$$

För att x-variablerna ska följa definitionen behöver bivillkor införas. Betrakta nu

$$x_{i,p,j,k} \geq \frac{1}{2}\left(f_{i,j,k} + f_{p,j,k+1} - 1\right),\ \forall i, p \in I,\ \forall j \in J,\ \forall k \in K \setminus \{H\}. \tag{8}$$

Detta bivillkor innebär att x-variablerna följer första delen av definitionen; om fält i och p körs i ordning k respektive $k+1$ på dag j så kommer $x_{i,p,j,k}$ vara lika med ett.[3] Däremot om en av eller båda termerna i högerledet i (8) är lika med noll så kommer $x_{i,p,j,k}$ kunna vara antingen noll eller ett. Enligt definitionen ska $x_{i,p,j,k}$ vara noll i detta fallet, vilket visas i ett kommande bivillkor när variabler för förflyttningar över natten införts, se (13). Följande bivillkor ser till att förflyttning mellan samma fält inte är möjlig:

$$x_{i,i,j,k} = 0,\ \forall i \in I,\ \forall j \in J,\ \forall k \in K \setminus \{H\}. \tag{9}$$

Bivillkoret gör att förflyttningar från fält i till i under en viss dag och ordning ej kan ske.

Det behövs en variabel som indikerar om ett fält är det sista fältet som lastas på en viss dag och i en viss ordning. Dessa variabler används för att modellera övergången mellan dagar. Vi introducerar därför

$$z_{i,j,k} = \begin{cases} 1, & \text{Om fält } i \text{ lastas på dag } j \text{ i ordning } k, \text{ och är sista fältet som lastas den dagen}, \\ & \forall i \in I,\ \forall j \in J,\ \forall k \in K, \\ 0, & \text{annars.} \end{cases}$$

Det första bivillkoret för denna variabel är att

$$z_{i,j,k} \geq\ f_{i,j,k} - \sum_{p=1}^{N} f_{p,j,k+1},\ \forall i \in I,\ \forall j \in J,\ \forall k \in K \setminus \{H\}. \tag{10}$$

Detta försäkrar att $z_{i,j,k}$ delvis uppfyller definitionen; om fältet är sist under dagen blir variabeln lika med ett.[4] Nu införs ytterligare ett bivillkor

$$\sum_{i=1}^{N}\sum_{k=1}^{H} z_{i,j,k} = 1,\ \forall j \in J, \tag{11}$$

vilket innebär att det finns endast ett fält som lastas sist varje dag. Därför är $z_{i,j,k}$ noll för alla fält och ordningar som inte beskriver det sista fältet för dagen. Med (10) och (11) följer z-variablerna definitionen.

Om ett fält har lastats klart men lastaren inte hinner flytta till nästa fält innan arbetsdagen är slut måste lastaren transporteras till nästa fält efter ordinarie arbetstid. Detta för att lastaren ska vara på plats och börja lasta direkt i början på nästa dag. En binär variabel införs som

$$u_{i,p,j} = \begin{cases} 1, & \text{om fält } i \text{ lastas sist på dag } j \text{ och fält } p \text{ lastas på dag } j+1 \text{ med ordning 1}, \\ & \forall i, p \in I,\ \forall j \in J \setminus \{D\}, \\ 0, & \text{annars,} \end{cases}$$

[3] Om både $f_{i,j,k}$ och $f_{p,j,k+1}$ är lika med ett (det vill säga vi åker från fält i till fält p på dag j vid ordning k), så måste $x_{i,p,j,k}$ vara ett.

[4] Om $\sum_{p=1}^{N} f_{p,j,k+1} = 0$ betyder det att det inte finns ett fält som lastas efter fält i. Därför ska $z_{i,j,k}$ vara lika med ett i detta fallet.

för att modellera förflyttningar efter arbetstid. Härnäst införs bivillkoret

$$u_{i,p,j} \geq \frac{1}{2}\left(f_{p,j+1,1} - 1 + \sum_{k=1}^{H} z_{i,j,k}\right), \ \ \forall i, p \in I, \ \ \forall j \in J \setminus \{D\}, \ \ \forall k \in K. \tag{12}$$

Detta bivillkor använder $f_{i,j+1,1}$ och $z_{i,j,k}$ så att variablerna $u_{i,p,j} = 1$ om fält i lastas sist på dag j och fält p lastas på dag $j+1$ med ordning ett.

Som tidigare nämnts så beskriver $x_{i,p,j,k}$ den rutt som lastaren transporteras under dagen, och $u_{i,p,j}$ beskriver ifall lastaren transporteras under natten. Då lastaren besöker totalt N stycken fält (totala antalet fält), måste lastaren flyttats $N-1$ gånger. Vi använder denna vetskap, tillsammans med definitionen av u-variablerna, för att introducera ett bivillkor som gör att x-variablerna följer definitionen från (7). Notera att $\sum_{i=1}^{N}\sum_{j=1}^{D}\sum_{k=1}^{H-1}\sum_{p=1}^{N} x_{i,p,j,k}$ är antalet förflyttningar som sker under dagen, och $\sum_{i=1}^{N}\sum_{p=1}^{N}\sum_{j=1}^{D-1} u_{i,p,j}$ är antalet förflyttningar som sker under kvällen. Totala antalet förflyttningar beskrivs av

$$\sum_{i=1}^{N}\sum_{j=1}^{D}\sum_{k=1}^{H-1}\sum_{p=1}^{N} x_{i,p,j,k} + \sum_{i=1}^{N}\sum_{p=1}^{N}\sum_{j=1}^{D-1} u_{i,p,j} - \sum_{i=1}^{N}\sum_{j=1}^{D-1} u_{i,i,j} = N - 1. \tag{13}$$

Vi subtraherar antalet gånger som lastaren stannar kvar på samma fält över natten, $\sum_{i=1}^{N}\sum_{j=1}^{D-1} u_{i,i,j}$, då detta inte räknas som en förflyttning. Totala antalet förflyttningar är nu definierad. Det medför att varje x-variabel som inte är lika med ett från (8) blir lika med noll. Därför följer x-variablerna sin definition. Tidigare inför vi bivillkor för att försäkra oss att definitioner av variabler följs. Vi visar i (32) att u-variablerna följer deras definition.

3.4 Modellering av tid och lastning

I detta avsnitt modelleras tidsapekter som arbetsdag och lastningstid för varje fält. Hänsyn tas också till mängden sockerbetor som lastas varje dag från varje fält. Det får inte planeras in mer arbete än vad det finns tid till på en dag. För att hantera detta introduceras en ny typ av variabel

$$\kappa_{i,j} = \text{antal timmar kvar att lasta på fält } i \text{ när dag } j \text{ tar slut}, \forall i \in I, \forall j \in J.$$

Antalet timmar kvar att lasta på ett fält är icke-negativ, vilket betyder att

$$\kappa_{i,j} \geq 0, \forall i \in I, \forall j \in J. \tag{14}$$

Om lastaren inte blir klar med det näst sista fältet för dagen, kan inte ett nytt fält schemaläggas efteråt samma dag. Därmed är $\kappa_{i,j}$ noll för alla fält utom det sista fältet varje dag. Lastaren måste lasta klart stukorna på alla fält innan det sista fältet för dagen:

$$\kappa_{i,j} \leq \widehat{M}\sum_{k=1}^{H} z_{i,j,k}, \ \ \forall i \in I, \forall j \in J. \tag{15}$$

Här är $\widehat{M}$ ett tal större än $\tau_{max} := \max_{i \in I} \tau_i$.[5] Det innebär att om fält i inte är sist på dag j, måste $\kappa_{i,j}$ vara noll. Det kan alltså finnas en restkvot av sockerbetor på sista fältet för dagen, men inte på fälten som lastats tidigare under dagen.

Med κ-variablerna tillåter vi att det sista fältet för dagen inte lastas klart. Detta används i följande bivillkor där vi ser till att det inte arbetas mer än tio timmar per dag:

$$\begin{aligned} &\sum_{i=1}^{N}\sum_{k=1}^{H} \tau_i f_{i,j,k} + \sum_{i=1}^{N}\sum_{p=1}^{N}\sum_{k=1}^{H-1} t_{i,p} x_{i,p,j,k} \\ &+ \sum_{i=1}^{N} \kappa_{i,j-1} - \sum_{i=1}^{N} \kappa_{i,j} - \sum_{i=1}^{N} \tau_i u_{i,i,j-1} + s_j = 10, \forall j \in J \setminus \{1\}. \end{aligned} \tag{16}$$

[5] Detta är ett så kallat big-M villkor, se avsnitt 2.1.1.

Vi motiverar nu detta bivillkor. Varje arbetsdag är maximalt 10 timmar. Bivillkoret måste se till att det inte går att schemalägga mer än 10 timmars arbete varje dag. Ett naivt sådant bivillkor är att

$$\text{tiden att lasta schemalagda fält} + \text{tiden att transportera lastaren mellan fälten} \leq 10.$$

Som det beskrivs i diskussionen kring κ-variablerna behöver möjligheten finnas att endast påbörja det sista fältet för dagen, och inte lasta klart det. Om ett schema innebär att ett fält delas på två dagar kommer bivillkoret ovan att brytas. Därför överförs en form av skuld till nästa dag. Lastaren kommer då behöva utföra både arbetet som schemalagts för dagen och det arbetet som inte blev klart dagen innan. Det naiva bivillkoret kan nu skrivas som

$$\begin{aligned}\text{tiden att lasta schemalagda fält} + \text{tiden att transportera lastaren mellan fälten} \\ +\text{skuld från dagen innan} \leq 10 + \text{skuld till nästa dag.}\end{aligned}$$

Genom att öka skulden som överförs till nästa dag ökar antalet timmar som kan schemaläggas den nuvarande dagen. Gårdagens skuld måste hanteras och adderas till den schemalagda tiden för nuvarande dagen. Slutligen önskas bivillkoret skrivas som en likhet, detta krävs för att värdet på κ-variablerna blir korrekta. Bivillkoret får då följande form:

$$\begin{aligned}\text{Tiden att lasta schemalagda fält} + \text{tiden att transportera lastaren mellan fälten} \\ +\text{skuld från dagen innan} + \text{tid kvar på dagen när vi är klara med dagens sista fält} \\ = 10 + \text{skuld till nästa dag.}\end{aligned}$$

Vi motiverar nu att denna formulering är ekvivalent med den given i (16). Följande summa beskriver tiden det tar att lasta alla de schemalagda fälten för en given dag j:

$$\sum_{i=1}^{N}\sum_{k=1}^{H}\tau_i f_{i,j,k}.$$

Notera att τ_i adderas om och endast om fält i är schemalagt på dag j, enligt definitionen av $f_{i,j,k}$. Den andra summan i (16),

$$\sum_{i=1}^{N}\sum_{p=1}^{N}\sum_{k=1}^{H-1} t_{i,p}x_{i,p,j,k},$$

är på samma sätt tiden det tar att transportera lastaren mellan de schemalagda fälten för dagen. Skulden från föregående dag, respektive skulden som överförs till nästa dag beskrivs med summorna:

$$\sum_{i=1}^{N}\kappa_{i,j-1}, \quad \text{respektive} \quad -\sum_{i=1}^{N}\kappa_{i,j}.$$

Notera att minustecknet framför den andra summan kommer från att denna term flyttats till vänsterledet.

Om ett fält lastas över två dagar adderas tiden det tar att lasta hela fältet i den första summan. Tiden som detta fält tar att lasta tas hänsyn till genom att använda κ. Därför adderas en sista summa till bivillkoret för att undvika dubbelräkning. I denna summa subtraheras τ_i om lastaren inte är på fältet för första gången:

$$-\sum_{i=1}^{N}\tau_i u_{i,i,j-1}.$$

Slutligen önskar vi ha en likhet i bivillkoret för att κ-variablerna ska ha de önskade egenskaperna. Därför inför vi variablerna

$$s_j = \text{tid kvar av arbetsdag } j \text{ då alla schemalagda fält är lastade}, \ \forall j \in J.$$

Tiden som är kvar av en arbetsdag är icke-negativ, därför är $s_j \geq 0$, för alla $j \in J$. Om vi adderar denna variabeln kan bivillkoret skrivas om till en likhet. För att se till att s_j följer definitionen måste två uppsättningar binära variabler, $\hat{s}_j$ och $\hat{\kappa}_j$, införas. Dessa variabler ser till att om $\sum_i^N \kappa_{i,j} > 0$ måste $s_j = 0$ och vice versa. Anledningen är att lastren inte både kan ha en skuld till nästa dag och ha tid kvar på arbetsdagen samtidigt. Bivillkoren

$$\frac{1}{\widetilde{M}} s_j \leq \hat{s}_j \leq \widetilde{M} s_j, \ \ \forall j \in J, \quad \text{och} \quad \frac{1}{\widehat{M}} \sum_{i=1}^{N} \kappa_{i,j} \leq \hat{\kappa}_j \leq \widehat{M} \sum_{i=1}^{N} \kappa_{i,j}, \ \ \forall j \in J, \tag{17}$$

ser till att $\hat{s}_j$ är lika med 1 om och endast om $s_j > 0$ och att $\hat{\kappa}_j$ är lika med 1 om och endast om $\sum_{i=1}^{N} \kappa_j > 0$. Här är $\widetilde{M}$ större än 10 som motsvarar en hel arbetsdag.[6] Detta tillsammans med bivillkoret

$$\hat{s}_j + \hat{\kappa}_j \leq 1, \ \ \forall j \in J, \tag{18}$$

innebär att s_j och $\kappa_{i,j}$ inte kan vara lika med ett samtidigt. Det kan därför inte finnas tid kvar på dagen, och en skuld i tid till nästa dag samtidigt. Notera att s_j och $\kappa_{i,j}$ tillåts vara noll samtidigt. Detta motsvarar fallet då lastaren blir klar med allt som är planerat för dagen utan att ha någon arbetstid kvar.

Sammantaget innebär detta att den matematiska formuleringen är ekvivalent med formuleringen i ord. På den första dagen finns det ingen skuld från tidigare dagar. Därmed kan bivillkoret för denna dag skrivas som:

$$\sum_{i=1}^{N} \sum_{k=1}^{H} \tau_i f_{i,1,k} + \sum_{i=1}^{N} \sum_{p=1}^{N} \sum_{k=1}^{H-1} t_{i,p} x_{i,p,1,k} - \sum_{i=1}^{N} \kappa_{i,1} + s_1 = 10. \tag{19}$$

Varje fält ska besökas minst en gång och om kvoten för fältet är så stor att den behöver delas upp på flera dagar stannar lastaren kvar på fältet över natten. Om lastaren stannar över natten på ett fält i mellan dag j och $j+1$, är $u_{i,i,j} = 1, \forall i \in I, \forall j \in J \setminus \{D\}$. Bivillkoret

$$\sum_{j=1}^{D} \sum_{k=1}^{H} f_{i,j,k} = 1 + \sum_{j=1}^{D-1} u_{i,i,j}, \ \ \forall i \in I, \tag{20}$$

beskriver hur många gånger under kampanjen som lastaren befinner sig på fält $i \in I$. Lastaren kommer befinna sig på varje fält minst en gång då sockerbetor på alla fält ska lastas och transporteras till sockerbruket. Då lastaren stannar kvar på ett fält mellan två dagar, befinner den sig på fältet båda dessa dagar, därav summan i högerledet.

Om det finns en skuld av tid för att lasta klart stukan på ett visst fält, måste detta fältet vara först i schemat på nästa dag. Detta beskriver vi med hjälp av bivillkoret

$$f_{i,j+1,1} \geq \frac{1}{\widehat{M}} \kappa_{i,j} - \frac{\upsilon}{\widehat{M}}, \ \ \forall i \in I, \ \ \forall j \in J \setminus \{D\}. \tag{21}$$

Bivillkoret tvingar $f_{i,j+1,1}$ att bli 1 om det finns en rest större än υ, det vill säga, när $k_{i,j} > \upsilon$. Vi inför $\upsilon = 0.245$ för att sätta en gräns för när det kan anses vara tidsmässigt fördelaktigt att lämna kvar sockerbetor på ett fält.[7] Om mängden sockerbetor som är kvar på fältet i slutet på dagen är mindre än vl kan inte lastaren befinna sig på fältet dagen efter, detta härleds i (26). Vidare får det inte finnas någon skuld större än υ från den sista dagen på ett givet fält. Detta beskrivs av bivillkoret

$$\kappa_{i,D} \leq \upsilon, \ \ \forall i \in I. \tag{22}$$

[6] Detta är ett till big-M villkor, se avsnitt 2.1.1.

[7] En lastbil kan lastas med 40 ton sockerbetor och lastningsfarten är 163 ton per timme. Det tar då cirka 0.245 timmar (15 minuter) att fylla en lastbil med 40 ton sockerbetor. Vi säger därför att om det endast krävs en lastbil för att lasta klart fältet efter dagen är slut, behöver dessa sockerbetor inte nödvändigtvis lastas och transporteras till sockerbruket. Om detta inte var fallet skulle lastaren vara tvungen att stanna på fältet över natten för att lasta en lastbil och sedan köra vidare till nästa fält. Vi gör alltså en avvägning för vilken mängd sockerbetor som kan anses vara försumbar.

För att vi ska kunna avgöra om ett fält har lastats klart införs följande variabler:

$$w_{i,j} = \text{Massan sockerbetor kvar på fält } i \text{ när dag } j \text{ påbörjas}, \ \ \forall i \in I, \forall j \in \{1, 2, ..., D, D+1\}.$$

Med dessa variabler visas hur mycket som är kvar att lasta på ett fält när en dag påbörjas. Med hjälp av (22) går det att härleda att, i slutet på kampanjen kan som mest vl ton sockerbetor på varje fält vara icke-levererade. Det kan ses som att dagen efter sista dagen på kampanjen ska mängden sockerbetor i stukorna vara mindre än eller lika med vl. Påståendet är ekvivalent med bivillkoret

$$w_{i,D+1} \leq vl, \ \ \forall i \in I. \tag{23}$$

Kvoten som är kvar i en stuka kan inte heller öka till en senare dag, därför är

$$w_{i,j} \geq w_{i,j+1}, \ \ \forall i \in I, \forall j \in J. \tag{24}$$

Vidare är mängden sockerbetor i början av dag ett på varje fält lika med skörden på fältet, vilket kan skrivas som

$$w_{i,1} = \psi_i, \ \ \forall i \in I. \tag{25}$$

För att undvika att lastaren besöker ett fält där det inte finns några betor kvar måste följande bivillkor för f-variablerna införas:

$$\sum_{k=1}^{H} f_{i,j,k} \leq \frac{w_{i,j}}{vl}, \ \ \forall i \in I, \ \ \forall j \in J. \tag{26}$$

Därmed försäkrar vi oss att lastaren endast befinner sig på ett fält där det finns mer än vl ton sockerbetor i början på dagen. Därmed kan lastaren inte transporteras tillbaka till ett fält den redan besökt. Tillsammans med tidigare bivillkor för u-variablerna innebär det att om lastaren är klar med ett fält när arbetsdagen tar slut måste lastaren flytta vidare till nästa fält samma kväll. På så sätt är lastaren redo att börja lasta direkt på morgonen dagen efter. Sammantaget betyder det att lastaren stannar kvar över natten på ett fält om och endast om det finns mer än vl sockerbetor.

Vi har nu beskrivit w-variablernas egenskaper och måste nu se till att dessa egenskaper följs. Detta görs genom rekursion, då differensen $w_{i,j} - w_{i,j+1}$ är mängden sockerbetor som har lastats på fält i under dag j. Denna mängd sockerbetor är lika med

$$w_{i,j} - w_{i,j+1} = l \left(\tau_i \sum_{k=1}^{H} f_{i,j,k} + \kappa_{i,j-1} - \kappa_{i,j} - \tau_i u_{i,i,j-1} \right), \ \ \forall i \in I, \ \ \forall j \in J \setminus \{1\}. \tag{27}$$

Termerna i högerledet motiveras på liknande vis som i (16).[8] På dag ett blir rekursionen

$$w_{i,1} - w_{i,2} = l \left(\tau_i \sum_{k=1}^{H} f_{i,1,k} - \kappa_{i,1} \right), \ \ \forall i \in I, \tag{28}$$

då det inte finns en skuld av tid från dagen innan, och lastaren inte stannar på ett fält från dagen innan.

3.5 Modellering av dagliga kvoten

Den dagliga kvoten q_j som sockerbruket vill ha in på dag j är ett exakt tal. Det är däremot inte säkert att det existerar ett schema som levererar exakt q_j sockerbetor för varje dag $j \in J$. Att införa ett bivillkor som tvingar lastaren att leverera en exakt kvot varje dag kan eventuellt medföra att problemet inte har någon tillåten lösning. Även om det existerar en tillåten lösning som levererar den korrekta mängden betor varje dag kan det vara en lösning som inte är önskvärd i verkligheten, till exempel en lösning med långa körsträckor mellan fälten. För att öka robustheten i modellen

[8] Termerna inom parentesen beskriver som i (16) den tid som lastaren lastar från stukan. Denna tid multipliceras med lastningsfarten l för att få den mängd sockerbetor som lastas under dagen.

införs två variabler som mäter hur mycket under, eller över, den exakta kvoten som levereras på dag j. Dessa variabler bidrar till en relaxering av q_j som gör att kvoten inte måste följas exakt. Vi introducerar, för varje $j \in J$, variablerna r_j och ρ_j, där r_j är hur många ton under den dagliga kvoten q_j som levereras, och ρ_j är hur många ton över den dagliga kvoten som levereras. Dessa variabler är icke-negativa, och därmed gäller det att

$$r_j \geq 0, \ \ \forall j \in J, \quad \text{samt} \quad \rho_j \geq 0, \ \ \forall j \in J. \tag{29}$$

Från beskrivningen av r_j och ρ_j ska maximalt en av de två variablerna vara noll-skild $\forall j \in J$, eftersom vi inte kan leverera både för mycket och för lite samtidigt. Variablerna associeras med kostnader i målfunktionen, när denna introduceras motiveras varför endast en av r_j och ρ_j kan vara noll-skild, se (32).

För att beräkna mängden sockerbetor som levereras under en dag tar vi fram ett bivillkor som relaterar skörden med f-variablerna. Notera att ett fält kan ha påbörjats på dag $j-1$ och ej lastats klart, därmed måste detta fält köras klart innan ett annat fält påbörjas. Därav subtraheras den eventuella mängden sockerbetor från stukan på fält i på dag j som inte hann levereras. Med relaxering av q_j genom r_j och ρ_j får vi därmed att

$$\sum_{i=1}^{N}\sum_{k=1}^{H} \psi_i f_{i,j,k} + l\sum_{i=1}^{N}(\kappa_{i,j-1} - \kappa_{i,j}) = q_j - r_j + \rho_j, \ \ \forall j \in J. \tag{30}$$

Summationen över i och k ger fälten som skördas på dag j. För första dagen på kampanjen gäller att

$$\sum_{i=1}^{N}\sum_{k=1}^{H} \psi_i f_{i,1,k} - l\sum_{i=1}^{N}\kappa_{i,1} = q_1 - r_1 + \rho_1, \tag{31}$$

då det inte finns någon skuld från dagen innan att beakta.

3.6 Målfunktion

Alla bivillkor för modellen är nu definierade och härnäst definieras målfunktionen. Målet är att minimera de rörliga kostnaderna för lastaren. Målfunktionen är därmed formulerad som

$$\sum_{i=1}^{N}\sum_{p=1}^{N}\sum_{j=1}^{D}\sum_{k=1}^{H-1} ct_{i,p}x_{i,p,j,k} + \sum_{j=1}^{D-1}\sum_{i=1}^{N}\sum_{p=1}^{N} ct_{i,p}u_{i,p,j} + \sum_{j=1}^{D}(\alpha r_j + \beta\rho_j)\,. \tag{32}$$

Den första summan beräknar den totala tiden lastaren kommer att transporteras under dagtid. Vi adderar tiden $t_{i,p}$ (tiden det tar att köra från fält i till fält j) om och endast om $x_{i,p,j,k} = 1$, vilket sker om och endast om vi kör från fält i till fält p. Detta multipliceras sedan med kostnaden c att köra lastaren per timme, för att få resultatet i kronor.

För att ta hänsyn till kostnaden för förflyttning av lastaren efter arbetstid inför vi den andra summan. Termen $c * t_{i,p}$ adderas om och endast om motsvarande u-variabel är lika med ett. Vi har i (12) sett till att alla u-variabler som ska vara lika med ett är det. För en optimal lösning kommer alla andra u-variabler att vara lika med noll, annars skulle kostnadsfunktionen generera ett större värde, ty alla termer i summan är icke-negativa. Det medför att i en optimal lösning kommer u-variablerna följa definitionen.

Den sista summan tar hänsyn till bestraffningarna för att leverera under eller över de dagliga kvoterna. Här införs straffparametrarna α och β som beskriver hur stor vikt modellen lägger vid att sockerbrukets kvot ska följas varje dag. Ett stort värde på α betyder att om det lastas för lite under en dag straffar det sig mer än för ett litet värde på α. På samma sätt gäller för värdet på β, men i fallet för om vi lastat för mycket. De införs som:

- α [SEK/ton], straffkonstant vid för liten leverans mot mängden q_j,
- β [SEK/ton], straffkonstant vid för för stor leverans mot mängden q_j.

Notera att valet på α och β har stor inverkan på hur den optimala lösningen ser ut. Valet av dessa parametrar reflekteras över i diskussionen.

I (30) och (31) ser vi att det är tillåtet att r_j och ρ_j är noll-skilda samtidigt, vilket tidigare nämnts inte ska vara möjligt. Vi motiverar nu varför r_j och $\rho_j, \forall j \in J$, inte kan vara noll-skilda samtidigt i en optimal lösning. Detta görs genom ett motsägelsebevis. Antag att det finns ett optimalt schema för vilket $r_j > 0$ och $\rho_j > 0$ för något $j \in J$. Variablerna ingår endast i ett bivillkor, nämligen (16) för dag j. I detta bivillkor gäller att ifall r_j reduceras kommer en reduktion av ρ_j av samma storlek att se till att bivillkoret fortfarande uppfylls. Detta går att göra fram tills dess att en av variablerna är lika med noll. Notera att i den sista summan i kostnadsfunktionen (32) är α och β större än noll. I en optimal lösning är r_j och ρ_j så små som möjligt, vilket innebär att en av de måste vara lika med noll. Vi kan därför säga att högst en av dem är noll-skild, då kostnadsfunktionen ska minimeras.

3.7 Sammanfattad modell

Nedan följer den fullständiga matematiska modellen för optimal transport av lastaren enligt modellbeskrivningen ovan:

$$\min \quad \sum_{i=1}^{N}\sum_{p=1}^{N}\sum_{j=1}^{D}\sum_{k=1}^{H-1} cht_{i,p}x_{i,p,j,k} + \sum_{j=1}^{D-1}\sum_{i=1}^{N}\sum_{p=1}^{N} cht_{i,p}u_{i,p,j} + \sum_{j=1}^{D}(\alpha r_j + \beta\rho_j),$$

sådant att

$$\sum_{i=1}^{N} f_{i,j,k} \leq 1, \ \forall j \in J, \ \forall k \in K,$$

$$\sum_{k=1}^{H} f_{i,j,k} \leq 1, \ \forall i \in I, \ \forall j \in J,$$

$$\sum_{p=1}^{N} f_{p,j,k+1} \leq \sum_{i=1}^{N} f_{i,j,k}, \ \forall j \in J, \ \forall k \in K \setminus \{H\},$$

$$x_{i,p,j,k} \geq \frac{1}{2}(f_{i,j,k} + f_{p,j,k+1} - 1), \ \forall i,p \in I, \ \forall j \in J, \ \forall k \in K \setminus \{H\},$$

$$x_{i,i,j,k} = 0, \ \forall i \in I, \ \forall j \in J, \ \forall k \in K \setminus \{H\},$$

$$z_{i,j,k} \geq \ f_{i,j,k} - \sum_{p=1}^{N} f_{p,j,k+1}, \ \forall i \in I, \ \forall j \in J, \ \forall k \in K \setminus \{H\},$$

$$\sum_{i=1}^{N}\sum_{k=1}^{H} z_{i,j,k} = 1, \ \forall j \in J,$$

$$u_{i,p,j} \geq \frac{1}{2}\left(f_{p,j+1,1} - 1 + \sum_{k=1}^{H} z_{i,j,k}\right), \ \forall i,p \in I, \ \forall j \in J \setminus \{D\},$$

$$\sum_{i=1}^{N}\sum_{j=1}^{D}\sum_{k=1}^{H-1}\sum_{p=1}^{N} x_{i,p,j,k} + \sum_{i=1}^{N}\sum_{p=1}^{N}\sum_{j=1}^{D-1} u_{i,p,j} - \sum_{i=1}^{N}\sum_{j=1}^{D-1} u_{i,i,j} = N - 1,$$

$$\kappa_{i,j} \geq 0, \ \forall i \in I, \ j \in J,$$

$$\kappa_{i,j} \leq \widehat{M}\sum_{k=1}^{H-1} z_{i,j,k}, \ \forall i \in I, \ \forall j \in J,$$

$$\sum_{i=1}^{N}\sum_{k=1}^{H} \tau_i f_{i,1,k} + \sum_{i=1}^{N}\sum_{p=1}^{N}\sum_{k=1}^{H-1} t_{i,p}x_{i,p,1,k} - \sum_{i=1}^{N}\kappa_{i,1} + s_1 = 10,$$

$$\sum_{i=1}^{N}\sum_{k=1}^{H} \tau_i f_{i,j,k} + \sum_{i=1}^{N}\sum_{p=1}^{N}\sum_{k=1}^{H-1} t_{i,p}x_{i,p,j,k} + \sum_{i=1}^{N}(\kappa_{i,j-1} - \kappa_{i,j}) - \sum_{i=1}^{N}\tau_i u_{i,i,j-1} + s_j = 10, \ \forall j \in J \setminus \{1\},$$

$$\frac{1}{\widetilde{M}} s_j \leq \hat{s}_j \leq \widetilde{M} s_j, \ \ \forall j \in J,$$
$$s_j \geq 0, \ \ \forall j \in J,$$
$$\frac{1}{\widehat{M}} \sum_{i=1}^{N} \kappa_{i,j} \leq \hat{\kappa}_j \leq \widehat{M} \sum_{i=1}^{N} \kappa_{i,j}, \ \ \forall j \in J,$$
$$\hat{s}_j + \hat{\kappa}_j \leq 1, \ \ \forall j \in J,$$
$$f_{i,j+1,1} \geq \frac{1}{\widehat{M}} \kappa_{i,j} - \frac{\upsilon}{\widehat{M}}, \ \ \forall i \in I, \ \ \forall j \in J \setminus \{D\},$$
$$\sum_{j=1}^{D} \sum_{k=1}^{H} f_{i,j,k} = 1 + \sum_{j=1}^{D-1} u_{i,i,j}, \ \ \forall i \in I,$$
$$\kappa_{i,D} \leq \upsilon, \ \ \forall i \in I,$$
$$w_{i,D+1} \leq \upsilon l, \ \ \forall i \in I,$$
$$w_{i,j} \geq w_{i,j+1}, \ \ \forall i \in I, \ \ \forall j \in J,$$
$$w_{i,1} = l\tau_i, \ \ \forall i \in I,$$
$$\sum_{k=1}^{H} f_{i,j,k} \leq \frac{w_{i,j}}{\upsilon l}, \ \ \forall i \in I, \ \ \forall j \in J,$$
$$w_{i,1} - w_{i,2} = l \left(\tau_i \sum_{k=1}^{H} f_{i,1,k} - \kappa_{i,1} \right), \ \ \forall i \in I,$$
$$w_{i,j} - w_{i,j+1} = l \left(\tau_i \sum_{k=1}^{H} f_{i,j,k} + \kappa_{i,j-1} - \kappa_{i,j} - \tau_i u_{i,i,j-1} \right), \ \ \forall i \in I, \ \ \forall j \in J \setminus \{1\},$$
$$\sum_{i=1}^{N} \sum_{k=1}^{H} \psi_i f_{i,1,k} - l \sum_{i=1}^{N} \kappa_{i,1} = q_1 - r_1 + \rho_1,$$
$$\sum_{i=1}^{N} \sum_{k=1}^{H} \psi_i f_{i,j,k} + l \sum_{i=1}^{N} (\kappa_{i,j-1} - \kappa_{i,j}) - l \sum_{i=1}^{N} \tau_i u_{i,i,j-1} = q_j - r_j + \rho_j, \ \ \forall j \in J,$$
$$\rho_j \geq 0, \ \ \forall j \in J,$$
$$r_j \geq 0, \ \ \forall j \in J.$$

4 Implementation och numeriska lösningar

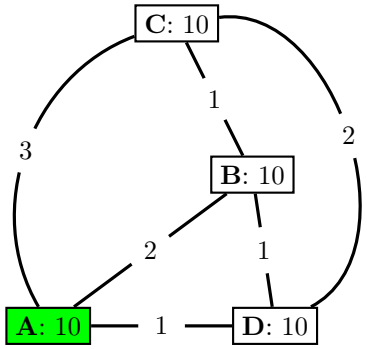
Grunden till lösare av linjära blandade heltalsprogrammeringsproblem baseras på Divide and conquer-algoritmer. Dessa algoritmer utmärker sig för dess förmåga att lösa stora komplexa problem. Algoritmen bryter ner ursprungsproblemet till mindre problem med liknande egenskaper. Detta gör algoritmen rekursivt tills den funnit ett problem som kan lösas. Ofta är dessa nedskalade problem basfall, eller så löses de med hjälp av en annan algoritm som gör arbetet snabbare. När den väl hittat lösningar till de mindre problemen, sätter den ihop delar av de lösningar den funnit, återigen rekursivt, tills den nått originalproblemet [4]. I arbetet implementeras och löses modellen i CPLEX som använder Branch and cut-algoritmen [7]. Detta är en typ av Divide and conquer-algoritm som beskrivs ovan. Verifikationen av modellen implementeras i IBM ILOG CPLEX Optimization Studio Version: 22.1.1.0 på en MacBook Pro 2020 med 8GB RAM och ett M1-chip samt CPLEX i Python 3.9 på en HP 15-dw1271no med 8GB RAM och Intel i5-10210U-processor. Implementationen av ett verkligt problem sker i IBM ILOG CPLEX Optimization. Båda implementeringar ger samma lösning när modellen verifieras. Det betyder att modellen är implementerad korrekt i IBM ILOG och Python. Vi väljer att inkludera ILOG-implementationen då koden är mer koncis. Vidare presenteras endast koden för det verkliga problemet, koden för scenario 1-4 är mycket lik och kan härledas från parametervärdena i A.2. Implementationen av det verkliga problemet finns i Appendix B.5 och B.6.

4.1 Verifikation av modellen

Modellen verifieras genom att den beräknar optimala scheman för ett antal scenarion som kan uppstå under en kampanj. Fördelen med att använda mindre scenarion är att vi enkelt kan räkna ut för hand vad den optimala lösningen ska vara. För scenario ett till fyra är lastningsfarten, transporthastigheten och kostnaden att transportera lastaren per kilometer lika med ett. Straffparametrarna α och β är lika med tre. Vidare är $\upsilon = 0.245$ för alla fyra scenarion. Viktigt att påpeka är att det kan finnas flera optimala lösningar till ett uppställt problem. Ett exempel är om stukan på sista fältet för dagen lastas klart innan arbetsdagen är slut. Lastaren kan förflyttas till nästa fält under eller efter arbetstid till samma kostnad. Detta måste tas hänsyn till då den implementerade lösningen endast ger ett svar, trots att det kan finnas flera optimala lösningar. Vi bestämmer att lastaren börjar på ett visst fält, för att sedan för hand beräkna den optimala lösningen, givet denna startpunkt.

4.1.1 Scenario 1

I detta scenario verifieras att modellen hittar den kortaste vägen mellan fälten. Vi har fyra fält, fyra dagar samt maximalt två ordningar. Fälten är benämnda [A, B, C, D] och motsvarande storlek på stukorna i ton är [10, 10, 10, 10] och varje dag ska tio ton sockerbetor levereras till sockerbruket. På grund av att det tar tio timmar att lasta varje fält kan inte lastaren befinna sig på mer än två fält per dag då lastningsfarten är lika med ett. Därför är det största ordningstalet lika med två. Parametervärdena ovan sammanfattas i A.2.1. Vi visualiserar denna uppställning i figur 2.

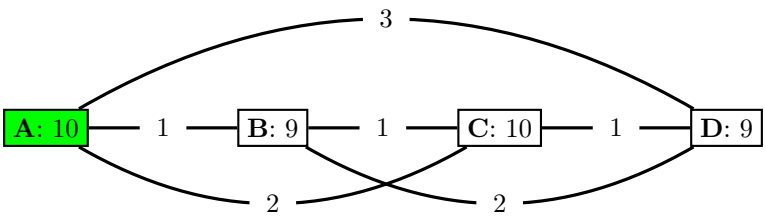


Figur 2: Scenario 1, nodernas värde är mängden sockerbetor i fältets stuka och kanternas värden är antal timmar det tar att transportera lastaren mellan fälten. Lastaren startar på fält A på dag ett, vilket indikeras genom att markera detta fält grönt.

Vi kan härleda från parameterbeskrivningen samt figur 2 att fälten ska lastas i ordning $[A, D, B, C]$, då det representerar den kortaste vägen. Ett fält och tio ton sockerbetor ska lastas per dag. Denna mängd sockerbetor är lika med den dagliga kvoten för varje dag. Detta måste vara den optimala lösningen, och är den lösning som fås när modellen implementeras. Värt att notera är att transporten av lastaren i detta scenario måste ske under natten, vilket implementationen av modellens ger.

4.1.2 Scenario 2

I detta scenario verifieras att lastaren transporteras på kvällen då ett fält är lastat och det inte finns tid kvar på arbetsdagen. Lastaren måste transporteras på kvällen för att börja lasta nästa fält följande morgon. Uppställningen är följande: på fyra dagar ska fyra fält lastas, vilka är i en följd med avstånd ett mellan angränsande fält. Fälten benämns [A, B, C, D] och motsvarande mängd sockerbetor i stukorna i ton är [10, 9, 10, 9]. Antalet ordningar kan begränsas till två då det minsta fältet har nio ton sockerbetor och transporttiden är en timme till närliggande fält. Det är därför inte möjligt för lastaren att befinna sig på mer än två fält under en arbetsdag. Mängden sockerbetor som sockerbruket behöver i ton för dag ett till fyra är [10, 9, 10, 9]. I A.2.2 sammanfattas parametervärdena ovan. Vi visualiserar denna uppställning i figur 3.

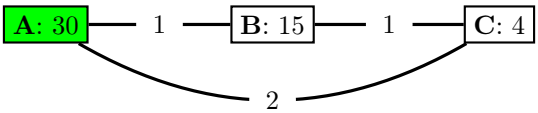


Figur 3: Scenario 2, nodernas värde är mängden sockerbetor i fältets stuka och kanternas värden är antal timmar det tar att transportera lastaren mellan fälten. Lastaren startar på fält A på dag ett, vilket indikeras genom att markera detta fält grönt.

Från beskrivningen av parametrarna kan en optimal lösning härledas. Den optimala lösningen är att lasta fälten i ordning A till D och att ett fält lastas per dag. Mängden sockerbetor som lastas varje dag är lika med kvoten sockerbruket behöver. Vidare ska lastaren transporteras mellan fält A till B och C till D under natten för att lastaren ska börja lasta direkt på morgonen påföljande dag. Transporten mellan fält B och C kan ske i slutet av dag två eller på natten mellan dag två och tre. Dessa är båda optimala scheman eftersom de är fullt fungerande, och gör totalt tre förflyttningar längs de kortaste vägarna. Lastaren kan inte göra färre förflyttningar än så, eller köra en mindre kostsam rutt. Den numeriska implementationen av modellen ger ovanstående lösning, med en förflyttning av lastaren mellan fält B och C på dagen. För att säkerställa att detta faktiskt är en optimal lösning varierar vi implementationen i detta scenario. Vi tvingar lastaren att köra mellan B och C under natten, och observerar målfunktionens värde. Det visar sig att kostnadsfunktionen har samma värde oavsett om transporten sker under dagen eller kvällen. Därmed är lösningen vi får optimal.

4.1.3 Scenario 3

Vi verifierar nu att att modellen kan hantera fält som måste delas upp på flera dagar på ett korrekt vis. Detta är ett viktigt scenario då det ofta sker under en kampanj. Vi verifierar att ett fält lastas klart innan ett nytt fält påbörjas. Att lastaren stannar kvar på fält över natten om det finns sockerbetor kvar verifieras också. Uppställningen är följande; på fem dagar ska tre fält lastas och maximalt tre ordningar är tillåtna. Maximalt antal ordningar motiveras med att det finns tre fält, och därför är inte ett större ordningstal nödvändigt. Transporttiden mellan angränsande fält är en timme. Mångden sockerbetor i stukorna i ton för fält A till C är [30, 15, 4]. Mängden sockerbetor som sockerbruket behöver dag ett till fem i ton är [10, 10, 10, 10, 9]. I A.2.3 sammanfattas parametervärdena ovan. Figur 4 visar grafiskt detta scenario.



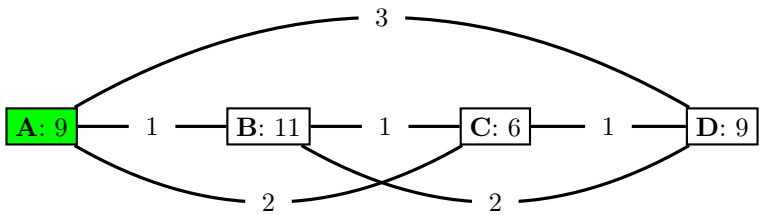
Figur 4: Scenario 3, nodernas värde är mängden sockerbetor i fältets stuka och kanternas värden är antal timmar det tar att transportera lastaren mellan fälten. Lastaren startar på fält A på dag ett, vilket indikeras genom att markera detta fält grönt.

Från beskrivningen av parametrarna kan en optimal lösning härledas. Fält A ska lastas från dag ett till tre. På dag tre lastaren transporteras under kvällen till fält B. Fält B lastas under dag fyra och delar av dag fem, och C lastas efter fält B under dag fem. Detta är det optimala schemat eftersom det är ett fungerande schema, och gör totalt två förflyttningar längs de kortaste vägarna.

Lastaren kan inte göra färre förflyttningar än så, vi kan inte heller välja billigare vägar och kvoten uppfylls exakt varje dag. Det är denna lösning som erhålls när modellen implementeras.

4.1.4 Scenario 4

I detta scenario ska flera aspekter verifieras. Först verifieras att modellen kan hantera att mängden sockerbetor på fälten inte är lika med den dagliga kvoten varje dag. Det verifieras också att lastaren kan bli klar tidigare ifall det inte finns mer arbete att utföra på en viss dag. Vi konstruerar detta scenario genom att på fyra dagar, där daglig kvot i ton är [8, 10, 8, 9], lasta fyra fält där mänden sockerbetor i stukorna på fälten i ton är [9, 11, 6, 9]. Maximalt antal ordningar bestäms till fyra, då det är det totala antalet fält. Vidare har angränsande fält avstånd ett mellan sig. Parametervärdena ovan sammanfattas i A.2.4. Vi visualiserar uppställningen i figur 5.



Figur 5: Scenario 4, nodernas värde är mängden sockerbetor i fältets stuka och kanternas värden är antal timmar det tar att transportera lastaren mellan fälten. Lastaren startar på fält A på dag ett, vilket indikeras genom att markera detta fält grönt.

Fälten ska levereras i ordningen $[A, B, C, D]$, där lastaren på dag ett lastar ett ton för mycket sockerbetor och avslutar dagen en timme för tidigt. Förflyttningen från fält A till B kan ske under dagen eller kvällen på dag ett. På andra dagen påbörjas lastningen av fält B och lastaren stannar kvar över natten för att på dag tre lasta ett sista ton sockerbetor. Senare på dagen ska lastaren köra till C, lasta klart, och sedan köra till D för att lasta ett ton. Lastaren ska stanna på fält D över natten mellan dag tre och fyra. På sista dagen levereras sedan ett ton för lite från fält D mot den dagliga kvoten till sockerbruket. I detta scenario är det omöjligt att varje dag uppfylla kvoten exakt, för om det endast lastas åtta ton betor första dagen så hinns det inte med att lasta tio ton nästa dag. Det kommer därför behöva lastas ett ton betor för mycket eller för lite någon dag. Eftersom det totala antalet sockerbetor är lika med den totala kvoten så måste det någon senare gång i schemat lastas ett ton betor för lite eller för mycket för att kompensera. Vi kan därför inte få ett totalt straff lägre än sex, vilket är exakt vad lösningen ger. Eftersom lastaren också kör kortaste vägen mellan fälten så måste detta vara en optimal lösning. Från scenario 2 vet vi att modellen ger samma kostnad om lastaren transporteras under dagen eller natten. Det finns därför två optimala lösningar till scenariot, lastaren kan antingen köra mellan fält A och B på dag ett eller under natten mellan dag ett och två. Den implementerade lösningen som erhålls är samma som beskrivs ovan och lastaren kör mellan fält A och B under dag ett.

4.2 Verkligt problem

Vi testar modellen på verklig data tillhandahållen av Nordic Sugar för säsongen 2021. I datan kan man se vilken odlare varje fält tillhör, hur stor skörden för fältet är och vilken *leveranskategori* som odlaren ingår i. Leveranskategorin bestämmer i vilka leveransperioder som odlaren ska lämna sockerbetor. Datan anger inte vilken leveransperiod som varje enskilt fält tillhör. Denna indelning gör vi själva, en beskrivning av detta ges i A.4.

När indelningen av fälten är gjord kan vi beräkna vektorer $\boldsymbol{\psi}$ för skörd, tiden $\boldsymbol{\tau}$ det tar att lasta stukorna på fälten och en matris t för tiden det tar att köra mellan fälten. Matrisen är uppbyggd så att element $t_{i,p}$ är den tid det tar för lastaren att köra mellan fält i och fält p. Då antalet fält, dagar och ordningar är ungefär lika för alla perioder valde vi godtyckligt leveransperiod ett.

I leveransperiod ett är antalet fält 47 och antalet dagar är 30. När vi använder verklig data är det svårare att välja ett bra maximalt värde på ordningstalet. När ordningstalet väljs är det viktigt att det inte är för litet. I så fall riskerar man att utesluta en optimal lösning. Om ordningstalet är större än nödvändigt kommer det ej att påverka vilken lösning som hittas. Alla möjliga lösningar för mindre ordningstal är fortfarande möjliga. Det är däremot fördelaktigt att inte använda ett för stort ordningstal, valet har en mycket stor inverkan på tidskomplexiteten. Den definitiva övre gränsen för alla problem är det totala antalet fält N, då lastaren inte kan vara på samma fält mer än en gång per dag.

För leveransperiod ett bedöms tio vara ett rimligt maximalt ordningstal. Detta motiveras i följande resonemang: Storleken på det maximala ordningstalet H är beroende av fältstorlek. Givet fälten som ingår i leveransperiod ett kan dessa sorteras i storleksordning. Vi beaktar då endast tiden τ_i det tar att lasta fält $i \in I$. Värdena på τ_i summeras sedan från minst till störst, tills en total ackumulerad tid om tio timmar uppnåtts. Det maximala ordningstalet är antalet element som adderats. Koden i B.5 implementerar denna metod. Ordningstalet där stämmer överens med det vi resonerat oss fram till.

Att inkludera hela leveransperiod ett resulterar i ett oerhört stort problem som vår begränsade datorkraft inte kan lösa. För att testa modellen på verklig data reducerar vi problemet till tio fält som ska lastas på fem dagar med största ordningstal fem. Mängden sockerbetor som ska levereras till sockerbruket varje dag har valts till 909 ton. Detta motiveras utifrån den totala mängden sockerbetor på fälten och att det varje dag ska levereras samma mängd till sockerbruket. Straffparametrarna α och β är lika med tio. Övriga parametrar är som i modellbeskrivningen. I A.3 sammanfattas parametervärdena ovan. Vi motiverar det maximala ordningstalet genom att undersöka data relaterad till problemet. De fem minsta fältens summerade mängd sockerbetor är 1051 ton. Mängden sockerbetor som ska lastas varje dag är 909 ton. På grund av straffparametrarna finns det därför aldrig någon anledning att lasta ett sjätte fält, då det medför en högre kostnad. Därmed är fem ett tillräckligt stort ordningstal. Det tar lösaren cirka 18 timmar att hitta en lösning till detta problem. Koden for implementeringen och lösningen kan ses i Appendix B.5 och B.6 Körtiden motiverar att det med vår datorkraft inte är möjligt att lösa motsvarande problem för en hel leveransperiod.

5 Samhälleliga och etiska aspekter

I detta avsnitt diskuteras etiska och samhälleliga aspekter, dels utifrån arbetets utförande, dels utifrån eventuella konsekvenser som arbetet medför. Då arbetet går ut på att ställa upp och optimera en matematisk modell, bedömmer vi att utförandet av arbetet inte innefattar några etiskt problematiska aspekter. Den data Nordic Sugar tillhandahåller innehåller odlares personuppgifter, därför har den hanterats varsamt. Exempelvis innehåller datan fältens geografiska position. I det verkliga problemet presenteras endast kvoten och tiden det tar att köra mellan fälten, för att ingen personlig information ska delas. All data och information som presenteras i rapporten har godkänts av Betfrakt och Nordic Sugar.

Verksamheten förlitar sig på att maskinförare kör lastarna. Det är därför viktigt att beakta huruvida schemaläggningen är rimlig för maskinförarnas hälsa. Exempelvis måste man diskutera hur stor del av transporten av lastaren som sker på kvällen. I modellen görs ingen skillnad på om lastaren transporteras under dagen eller natten. Om schemat läggs så att en stor del av transporterna mellan fält körs på kvällen, kan maskinarbetarnas uppskattade arbetsbelastning med övertid öka. Detta är ett problem som kan beaktas i framtida modellering genom att införa en gräns för hur mycket transport som får ske på kvällen. Betfrakt bestämmer i samråd med maskinförarna en timlön baserat på mängden kvällsarbete som kan uppstå under kampanjen. Två stycken maskinförare delar på en lastare under arbetsveckan, och arbetar antingen tre eller fyra dagar i veckan vardera. Detta arbetssätt existerar eftersom arbetstiden är cirka tio timmar per dag och maskinförarna ska få tillräckligt med vila. Då arbetet fokuserar på historisk data med ett fixt antal fält och dagar, kommer inte arbetsbördan öka eller minska markant genom att införa optimering. Viktigt att påpeka är att lösningen endast blir optimalt för den matematiska modellen och att ett beräknat schema måste verifieras så att det funkar praktiskt och är rättvist för alla parter.

Idag drivs lastarna med fossila bränslen vilket medför att miljöpåverkan är intressant att diskutera. Givet kvoten sockerbetor som sockerbruket behöver och lastarens utsläpp är den mest kostnadseffektiva rutten också den med minst miljöpåverkan. Detta eftersom den kortaste transportsträckan kräver mindre drivmedel och tid att köra. Lastarna förbrukar 20 liter diesel per timme vid transport på väg och kör med en hastighet på 20 kilometer per timme. Lastarna har därför en bränsleförbrukning på 10 liter diesel per mil. Detta kan jämföras med en generell pickup, exempelvis en Toyota Hilux, som förbrukar 0.87 liter diesel per mil [6]. Lastaren förbrukar alltså cirka elva gånger så mycket diesel som en Toyota Hilux. Om optimeringsverktyget används på resterande områden kan miljöpåverkan för leveranserna av sockerbetor i Skåne minska.

6 Diskussion

Det huvudsakliga målet med arbetet har varit att ta fram en matematisk modell som beskriver schemat för hur lastaren ska köra mellan fält. Modellen har tagits fram från en beskrivning av verksamheten med vissa antaganden som diskuteras i denna text. Vidare har modellen verifierats med fyra scenarion. I dessa scenarion gav lösaren till modellen exakt samma svar som lösningen beräknad för hand. Modellen har implementerats i två olika lösare oberoende av varandra. Implementeringarna har gjorts korrekt eftersom de gav samma svar för de fyra scenarierna i avsnitt 4.1.

Körtiden beror bland annat på lösningsalgoritmen. Algoritmen ”Branch and cut” har använts, som i värsta fall har en tidskomplexitet av $O(2^N)$, där N är antalet binära variabler. Detta grundar sig i att algoritmen delar upp problemet i flera små mindre problem som kallas noder. Varje binär variabel ger upphov till två möjligheter, antingen är den noll eller ett. Detta resulterar i 2^N möjliga noder. I värsta fall måste problemen i alla noder lösas, vilket leder till den exponentiella tidskomplexiteten. Viktigt att notera är att ”Branch and cut” faktiska tidskomplexitet är oerhört problemberoende, och kan variera kraftigt. Detta är alltså en grov uppskattning, men förklarar till viss del den långa körtiden. Modellen består av ett stort antal variabler för en leveransperiod, som omfattas av 47 fält som ska lastas under 30 dagar med största ordningstal 10, se avsnitt 4.2. Om vi endast undersöker x-variablerna, kommer de ge upphov till $47 \cdot 47 \cdot 30 \cdot 9$ variabler, vilket är mer än 500 000. I avsnitt 4.2 begränsades ett verkligt scenario till 10 fält, 5 dagar med största ordningstal 5. Körtiden för detta scenario var 18 timmar, vilket tyder på att vår datorkraft inte är tillräcklig för att hitta ett optimalt schema för en hel leveransperiod. Modellen kan egentligen endast verifieras mot de fyra scenarierna från avsnitt 4.1. En kampanj byggs upp av sammansättningar av de scenarion som modellen verifierats med. Då modellen har klarat av att hitta optimala lösningar till uppställningar som kan förekomma under en kampanj, ger det förhoppning att modellen ger ett optimalt schema för en hel leveransperiod alternativt kampanj.

Vidare är en intressant aspekt hur värdena på straffparametrarna α och β bestäms. Storleken på parametrarna indikerar hur modellen straffar avvikelser från de dagliga kvoterna. Sockerbruket kan exempelvis ha god möjlighet att lagra mer sockerbetor än den dagliga kvoten, vilket resulterar i ett lägre värde på β. Eventuellt innebär en leverans mindre än den dagliga kvoten stora kostnader för sockerbruket. Hur stor denna kostnaden är påverkar värdet på α. I alla scenarion presenterade i rapporten är värdena för α och β lika, vilket antar att det är lika kostsamt för sockerbruket att få in för mycket som för lite sockerbetor. I verkligheten är detta nödvändigtvis inte fallet. För att skapa en verklighetstrogen modell behöver därför kostnaderna för avvikelser från de dagliga kvoterna undersökas. På så sätt kan motiverade värden på α och β bestämmas.

Om man jämför arbetet med tidigare tankar inom optimal schemaläggning skiljer det sig i modellernas uppbyggnad. Vanligtvis brukar modeller byggas upp med hjälp av ett villkorsprogram, vilket innefattar att hitta optimala lösningar till kombinatoriska problem. Ett schema kan byggas upp genom att hitta en kombination av ordningar som platser besöks [7]. Vi byggde i stället upp modellen som ett linjärt blandat heltalsproblem, då vi har mer erfarenhet av denna typ av modellering. Vi kan däremot inte avgöra om en modell härledd med villkorsprogrammering hade haft kortare körtid. En sådan modell kräver färre variabler, men är icke-linjär. Detta tyder på att lösningsalgoritmen för denna modell hade varit mer tidskrävande.

Gällande framtida arbete inom området kan modellen betraktas som ett ramverk för modeller av andra men snarlika typer av schemaläggning. Ett förbättringsområde för modellen är att uppnå en bättre tidskomplexitet. Förhoppningen är då att ett optimalt schema för en leveransperiod kan erhållas, trots begränsad datorkraft. I framtiden kan även verktyg utvecklas för att tolka variabler till ett visuellt schema.

Avslutningsvis kan modellen användas som ett komplement till dagens manuella schemaläggning. Att använda modellen för att skapa ett schema som sedan revideras av ansvariga i leveransgrupperna utifrån deras erfarenhet bedöms vara en god strategi för att schemalägga en kampanj.

Referenser

[1] Andréasson N, Evgrafov A, Patriksson M, et al. An Introduction to Continous Optimization. 3:e uppl. Lund: Studentlitteratur: 2019. 1, Modelling and classification, sid. 14-20.

[2] Gurobi Optimization. Dealing with big-M constraints [Internet]. Gurobi Optimization, LLC. Hämtad 2023-04-20. Tillgänglig via: `https://www.gurobi.com/documentation/9.5/refman/dealing_with_big_m_constra.html`.

[3] Dantzig G, Thapa M. Linear Programming 1: Introduction. 1:a uppl. New York, NY: Springer New York: 1997. Preface; sid. xxxi.

[4] Vazirani U. Divide-and-conquer algorithms [Internet]. Berkeley: Umesh V. Vazirani; 2006 [hämtad 2023-03-01]. Tillgänglig via: `https://people.eecs.berkeley.edu/~vazirani/algorithms/chap2.pdf`.

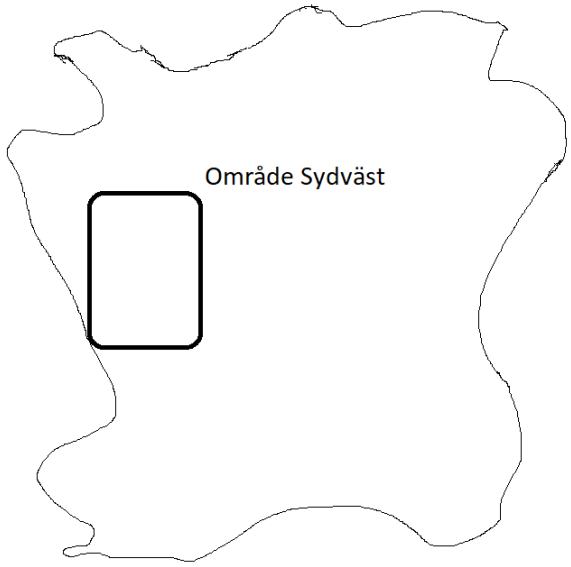
[5] IBM. Branch and cut in CPLEX [Internet]. The International Business Machines Corporation; 2021 [hämtad 2023-05-09]. Tillgänglig via: `https://www.ibm.com/docs/en/icos/12.10.0?topic=concepts-branch-cut-in-cplex`.

[6] Toyota. Hilux [Internet]. Toyota. Hämtad 2023-05-03. Tillgänglig via: `https://www.toyota.se/bilar/hilux`.

[7] IBM. Why constraint programming? [Internet]. The International Business Machines Corporation; 2021 [hämtad 2023-05-09]. Tillgänglig via: `https://www.ibm.com/docs/en/icos/12.8.0.0?topic=programming-why-constraint`.

A Appendix 1 – Teori

A.1 Karta av Skåne



Figur 6: En grov skiss av Skåne. Område Sydvästs ungefärliga gränser är utmarkerat.

A.2 Parametervärden scenario 1-4

A.2.1 Parametervärden scenario 1

$I = \{A, B, C, D\}$	$J = \{1, 2, 3, 4\}$	$K = \{1, 2, 3\}$
$c = 1$	$h = 1$	$l = 1$
$\alpha = 3$	$\beta = 3$	$\upsilon = 0.245$
$\mathbf{q} = [10, 10, 10, 10]$	$\boldsymbol{\psi} = [10, 10, 10, 10]$	$\boldsymbol{\tau} = [10, 10, 10, 10]$

$$t = \begin{pmatrix} 0 & 2 & 3 & 1 \\ 2 & 0 & 1 & 1 \\ 3 & 1 & 0 & 2 \\ 1 & 1 & 2 & 0 \end{pmatrix}$$

A.2.2 Parametervärden scenario 2

$I = \{A, B, C, D\}$	$J = \{1, 2, 3, 4\}$	$K = \{1, 2, 3\}$
$c = 1$	$h = 1$	$l = 1$
$\alpha = 3$	$\beta = 3$	$\upsilon = 0.245$
$\mathbf{q} = [10, 9, 10, 9]$	$\boldsymbol{\psi} = [10, 9, 10, 9]$	$\boldsymbol{\tau} = [10, 9, 10, 9]$

$$t = \begin{pmatrix} 0 & 1 & 2 & 3 \\ 1 & 0 & 1 & 2 \\ 2 & 1 & 0 & 1 \\ 3 & 2 & 1 & 0 \end{pmatrix}$$

A.2.3 Parametervärden scenario 3

$I = \{A, B, C\}$	$J = \{1, 2, 3, 4, 5\}$	$K = \{1, 2, 3\}$
$c = 1$	$h = 1$	$l = 1$
$\alpha = 3$	$\beta = 3$	$\upsilon = 0.245$
$\mathbf{q} = [10, 10, 10, 10, 9]$	$\boldsymbol{\psi} = [30, 15, 4]$	$\boldsymbol{\tau} = [30, 15, 4]$

$$t = \begin{pmatrix} 0 & 1 & 2 \\ 1 & 0 & 1 \\ 2 & 1 & 0 \end{pmatrix}$$

A.2.4 Parametervärden scenario 4

$I = \{A, B, C, D\}$	$J = \{1, 2, 3, 4\}$	$K = \{1, 2, 3\}$
$c = 1$	$h = 1$	$l = 1$
$\alpha = 3$	$\beta = 3$	$\upsilon = 0.245$
$\mathbf{q} = [8, 10, 8, 9]$	$\boldsymbol{\psi} = [9, 11, 6, 9]$	$\boldsymbol{\tau} = [9, 11, 6, 9]$

$$t = \begin{pmatrix} 0 & 1 & 2 & 3 \\ 1 & 0 & 1 & 2 \\ 2 & 1 & 0 & 1 \\ 3 & 2 & 1 & 0 \end{pmatrix}$$

A.3 Parametervärden till det verkliga problemet

$I = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$	$J = \{1, 2, 3, 4, 5\}$	$K = \{1, 2, 3, 4, 5\}$
$c = 1300$	$h = 20$	$l = 163$
$\alpha = 10$	$\beta = 10$	$\upsilon = 0.245$
$\mathbf{q} = [909, 909, 909, 909, 909]$		
$\boldsymbol{\psi} = [574.08, 467.71, 680.56, 319.8, 127.41, 86.22, 168.61, 703.05, 350.56, 1067.2]$		
$\boldsymbol{\tau} = [3.522, 2.8694, 4.1752, 1.962, 0.78168, 0.52896, 1.0344, 4.3132, 2.1507, 6.5472]$		

$$t = \begin{pmatrix} 0 & 0.21552 & 0.17955 & 0.62272 & 0.93097 & 0.91266 & 0.93611 & 0.37245 & 0.46027 & 0.42666 \\ 0.21552 & 0 & 0.037448 & 0.56792 & 1.0776 & 1.0553 & 1.0884 & 0.17802 & 0.63814 & 0.44823 \\ 0.17955 & 0.037448 & 0 & 0.58125 & 1.058 & 1.0364 & 1.0677 & 0.20298 & 0.61136 & 0.44648 \\ 0.62272 & 0.56792 & 0.58125 & 0 & 0.85099 & 0.81992 & 0.87894 & 0.68244 & 0.659 & 0.24277 \\ 0.93097 & 1.0776 & 1.058 & 0.85099 & 0 & 0.032023 & 0.037505 & 1.2551 & 0.48557 & 0.71071 \\ 0.91266 & 1.0553 & 1.0364 & 0.81992 & 0.032023 & 0 & 0.067333 & 1.2325 & 0.47252 & 0.68269 \\ 0.93611 & 1.0884 & 1.0677 & 0.87894 & 0.037505 & 0.067333 & 0 & 1.2662 & 0.48495 & 0.73165 \\ 0.37245 & 0.17802 & 0.20298 & 0.68244 & 1.2551 & 1.2325 & 1.2662 & 0 & 0.81429 & 0.60613 \\ 0.46027 & 0.63814 & 0.61136 & 0.659 & 0.48557 & 0.47252 & 0.48495 & 0.81429 & 0 & 0.42606 \\ 0.42666 & 0.44823 & 0.44648 & 0.24277 & 0.71071 & 0.68269 & 0.73165 & 0.60613 & 0.42606 & 0 \end{pmatrix}$$

A.4 Detaljerad diskussion kring indelning av fält

Indelningen av fälten i de olika leveransperioderna görs med avseende på vilken *leveranskategori* som odlaren tillhör. Leveranskategorin beskriver antalet tillfällen och i vilka leveransperioder odlaren ska lämna sockerbetor i. Antalet leveranstillfällen odlaren har beror på storleken av fälten. En odlare med mer sockerbetor blir uppdelad i fler perioder än en odlare med mindre. Sockerbetor är bland annat känsliga för kyla som kan leda till att de ruttnar, vilket är en stor risk under den senare delen av kampanjen. Det finns därför ett rotationssystem av leveranskategorier över fem år. Detta innebär att odlarens leveranskategori skiftas varje år för att skapa rättvisa förhållanden. Indelningen av odlare i leveranskategorier för år 2021 är gjord av Nordic Sugar och tillhandahålls arbetet.

Datan från Nordic Sugar innehåller vilken odlare varje fält tillhör, fältens skörd och vilken leveranskategori som odlaren är en del av. Det är från denna data vi delar in fälten i leveransperioder, med hjälp av koden i B.1. När indelningen görs måste vi se till att vi följer odlarenas leveranskategori, och att fördelningen av total mängd sockerbetor blir så jämn som möjligt över leveransperioderna. Detta är viktigt då sockerbrukets behöver en jämn mängd sockerbetor varje dag under kampanjen.

Vissa odlare har färre fält än antal leveranstillfällen. Detta kan ske eftersom indelningen i leveranskategorier beror på totala antalet hektar som odlaren har. En odlare med stora men få antal fält kan behöva leverera fler gånger än antalet fält den har. I detta fall delar vi odlarens största fält i två lika stora delar rekursivt. Detta görs tills odlaren har nog med fält för att täcka alla sina leveranstillfällen. Om en odlare istället har fler fält än antal leveranstillfällen kommer fälten delas in så jämnt som möjligt.

När en odlares fält sedan ska delas in i de olika leveransperioderna så ser koden i B.1, till att placera odlarens största fält i den leveransperiod som för tillfället har minst antal areal. På detta vis blir fördelningen av areal mellan leveransperioderna relativt jämn.

När alla fält är indelade i leveransperioder kan tiden det tar att köra mellan dessa fält beräknas. Varje fälts koordinater är givna i datan från Nordic Sugar, vilket gör att vi kan räkna ut avståndet mellan varje fält i perioden. Med hjälp av lastarens transporthastighet kan tiden det tar att köra mellan varje par av fält beräknas. Detta görs i koden i B.2, och resultatet är en matris. I vårt arbete benämner vi denna matris som t.

B Appendix 2 – Källkod

B.1 Indelning av fält

Listing 1: Python-kod för indelning av fält

```
import pandas as pd
import csv

# Read and store content
# of an excel file

read_file = pd.read_excel ("dat_sockerbetor.xlsx")

# Write the dataframe object
# into csv file
read_file.to_csv ("kand_csv.csv",
                  index = None,
                  header=True)

list_all_data = [] #list of all data from csv

with open("./kand_csv.csv", 'r') as file:
    csvreader = csv.reader(file)
    for row in csvreader:
        list_all_data.append(row)
file.close()

list_s = [] #list of values in Syd

for row in list_all_data:
    if row[2] == '401701':
        list_s.append(row)

#dictionary of categories and their respective leveransperioder
dict_categories = {11:[1], 12:[3], 13:[5], 14:[2], 15:[4],
21:[1,3], 22:[2,4], 23:[3,5], 24:[1,4], 25:[2,5],
31:[1,3,5], 32:[1,2,4], 33:[2,3,5], 34:[1,3,4], 35:[2,4,5],
41:[1,3,4,5], 42:[1,2,4,5], 43:[1,2,3,5], 44:[1,2,3,4], 45:[2,3,4,5],
50:[1,2,3,4,5], 60:[1,2,3,4,5,6]}

#Make a dictionary with key bonde value leveransgrupp
d_farmer_cat = {}
for row in list_s:
    if row[0] not in d_farmer_cat.keys():
        d_farmer_cat.update({row[0]: row[8]})

#Make a dictionary with key bonde value field
d_farmer_field = {}
for row in list_s:
    if row[0] not in d_farmer_field.keys():
        d_farmer_field.update({row[0]: [row[1]]})
    else:
        d_farmer_field[row[0]].append(row[1])

#Make a dictionary with key field value harvest
d_field_q = {}
for row in list_s:
    d_field_q.update({row[1]: [row[7], row[3], row[4]]})



d_period = {1:[], 2:[], 3:[], 4:[], 5:[], 6:[]}
#The dictionary says which fields are in which leveransperiod

def splitfield(farmer, fieldnr,o):
    newfieldnr = int(fieldnr) + 1000000*(2)^o + o
    newsize = float(d_field_q[fieldnr][0]) / 2
    d_field_q[fieldnr] = [float(d_field_q[fieldnr][0]) / 2, float(d_field_q[fieldnr][1]),
    float(d_field_q[fieldnr][2])] #Halve yield of the old biggest one
    d_farmer_field[farmer].append(newfieldnr)
    d_field_q.update({newfieldnr: [newsize, d_field_q[fieldnr][1], d_field_q[fieldnr][2]]})

def addfield(fieldnr, period):
    d_period[period].append(fieldnr) #add to field
    total_q_period[period] = total_q_period[period] + float(d_field_q[fieldnr][0]) #add to total harvest


#Step 0: Given leveransgrupp , number of apperances

def numdel(category):
    value = int(category)//10
    return value
```

```
def check_split(i):
#Step 1: Find which farmers need to have a field split
    split_farmers = []
    for farmer in d_farmer_cat.keys():
        if len(d_farmer_field[farmer]) < numdel(d_farmer_cat[farmer]):
            split_farmers.append(farmer)


#Step 2: Split the biggest fields where it is needed
    if split_farmers == []:
        return True

    for farmer in split_farmers:
        list_q = []
        list_fields = []
        for j in d_farmer_field[farmer]:
            list_fields.append(j)
            list_q.append(float((d_field_q[j][0])))
        split_val = max(list_q)
        split_index = list_q.index(split_val)
        nr = list_fields[split_index] #The ID of the field to split
        splitfield(farmer, nr, i)
    check_split(i+1)

check_split(0) #Splits everything nicely!

vist = []
for k,v in d_farmer_field.items():
    for element in v:
        if element in vist:
            print(element)
        else:
            vist.append(element)


#Step 3: The rest

total_q_period = {1:0, 2:0, 3:0, 4:0, 5:0, 6:1000000}
for farmer in d_farmer_cat.keys():
    if len(d_farmer_field[farmer]) == numdel(d_farmer_cat[farmer]): # If exactly the same, just split one to each
        category = int(d_farmer_cat[farmer]) #Leveransgrupp
        deliveries = dict_categories[category]
        for i in range(0,numdel(category)):
            period = deliveries[i]
            addfield(d_farmer_field[farmer][i], period)


    else: #if more fields than periods
        period_q = [0, 0, 0, 0, 0, 0]
        d_field_holder = {} #Dictionary of fields, harvest for a farmer
        for i in range(0, len(d_farmer_field[farmer])):
            #List of fields,harvest
            d_field_holder.update({d_farmer_field[farmer][i]: d_field_q[d_farmer_field[farmer][i]]})
        for i in d_field_holder:
            sorted_period = sorted(total_q_period.items(), key=lambda x: x[1])
            current_period = sorted_period[0][0] #Period we should add to
            levgrupp = d_farmer_cat[farmer]
            allowed_periods = dict_categories[float(levgrupp)] #check allowed periods
            k = 0
            while k < 100:
                if current_period in allowed_periods:
                    if period_q[current_period-1] == 0: # If we have not been here this period
                        addfield(i, current_period) #add
                        period_q[current_period-1] = 1
                        k = 200 #exit loop
                    elif sum(period_q) == len(allowed_periods):
                        addfield(i, current_period)
                        period_q[current_period -1] = 1
                        k = 200
                    else:
                        current_period = sorted_period[k+1][0]
                        k = k +1
                else:
                    current_period = sorted_period[k+1][0]
                    k = k + 1


#helper function csv for fields and their info
def helper_fields_csv(i):
    period_holder = []
    for k in d_period[i]:
        period_holder.append([k,d_field_q[k][0]])
    return period_holder
```

```
#Write csv of fields and info for each period
with open('field_1.csv', 'w', encoding='UTF8', newline='') as f:
    writer = csv.writer(f)

    writer.writerows(helper_fields_csv(1))


with open('field_2.csv', 'w', encoding='UTF8', newline='') as f:
    writer = csv.writer(f)

    writer.writerows(helper_fields_csv(2))


with open('field_3.csv', 'w', encoding='UTF8', newline='') as f:
    writer = csv.writer(f)

    writer.writerows(helper_fields_csv(3))


with open('field_4.csv', 'w', encoding='UTF8', newline='') as f:
    writer = csv.writer(f)

    writer.writerows(helper_fields_csv(4))


with open('field_5.csv', 'w', encoding='UTF8', newline='') as f:
    writer = csv.writer(f)

    writer.writerows(helper_fields_csv(5))

#Write csv of fields for each period
with open('period_1.csv', 'w', encoding='UTF8', newline='') as f:
    writer = csv.writer(f)

    writer.writerow(d_period[1])

with open('period_2.csv', 'w', encoding='UTF8', newline='') as f:
    writer = csv.writer(f)

    writer.writerow(d_period[2])

with open('period_3.csv', 'w', encoding='UTF8', newline='') as f:
    writer = csv.writer(f)

    writer.writerow(d_period[3])

with open('period_4.csv', 'w', encoding='UTF8', newline='') as f:
    writer = csv.writer(f)

    writer.writerow(d_period[4])

with open('period_5.csv', 'w', encoding='UTF8', newline='') as f:
    writer = csv.writer(f)

    writer.writerow(d_period[5])
```

B.2 Beräkning transporttid mellan fält

Listing 2: Python-kod för beräkning av transporttid mellan fält.

```
from indelning import *
import math

def distance_fields(lat_1, lat_2, lon_1, lon_2):
    '''
    Compute distance between two stops based on their respective longitude and
    latitude.
    Input: longitude and latitude for stop1 and stop2. Output: distance in
    kilometers
    '''
    #calculate longitude and latitude given with radians instead of degrees
    lat_1 = lat_1*math.pi/180; lat_2 = lat_2*math.pi/180
    lon_1 = lon_1*math.pi/180; lon_2 = lon_2*math.pi/180

    r = 6371.009 #radii earth in kms

    lat_m = (lat_1 + lat_2)/2 #average val lat
    delta_lat = lat_1 - lat_2; delta_lon = lon_1 - lon_2 #change in lat and lon

    d = math.sqrt((delta_lat)**2 + (math.cos(lat_m)*(delta_lon))**2)*r
```

```
    return d

dict_fields_info = d_field_q #fields and list of their info

dict_groups = d_period #leveransgrupp and list of fields

h = 20 #speed machine km/h
dict_group_time = {1:[], 2:[], 3:[], 4:[], 5:[], 6:[]} #hours
#create matrix of time between field i and j for all combinations of i and j
#for every period
for group, fields in dict_groups.items():
    for i in range(len(fields)):
        distance_holder = []
        for j in range(len(fields)):
            lon1 = float(dict_fields_info[fields[i]][1])
            lat1 = float(dict_fields_info[fields[i]][2])
            lon2 = float(dict_fields_info[fields[j]][1])
            lat2 = float(dict_fields_info[fields[j]][2])

            d = distance_fields(lat1, lat2, lon1, lon2)
            distance_holder.append(d/h)

        dict_group_time[group].append(distance_holder)


#Write a csv file for each period, shaped as matrix
with open('time_1.csv', 'w', encoding='UTF8', newline='') as f:
    writer = csv.writer(f)

    writer.writerows(dict_group_time[1])

f.close()

with open('time_2.csv', 'w', encoding='UTF8', newline='') as f:
    writer = csv.writer(f)

    writer.writerows(dict_group_time[2])

f.close()

with open('time_3.csv', 'w', encoding='UTF8', newline='') as f:
    writer = csv.writer(f)

    writer.writerows(dict_group_time[3])

f.close()

with open('time_4.csv', 'w', encoding='UTF8', newline='') as f:
    writer = csv.writer(f)

    writer.writerows(dict_group_time[4])

f.close()

with open('time_5.csv', 'w', encoding='UTF8', newline='') as f:
    writer = csv.writer(f)

    writer.writerows(dict_group_time[5])

f.close()
```

B.3 Big-M och största ordningstal

Listing 3: Matlab-kod som beräknar big-M värde och största ordningstal.

```
clearvars; clc; %Clear previous runs.

%Choose what case you want to run.
choice = input('Choose what case you want to run: ');

switch choice
    case 1 %Case 1:
        tau = sort([10 9 10 9]);
    case 2 %Case 2:
        tau = sort([10 10 10 10]);
    case 3 %Case 3:
        tau = sort([30 15 4]);
    case 4 %Case 4:
        tau = sort([9 11 6 9]);
    case 5 %Case 5:
        %Extracts the ten first fields from data and sorts it by size [tonnes].
```

```
        data = readmatrix("field_1.txt"); data = data(1:10, 2);
        prognosis = sort(data);
        l = 163; %Loading speed [ton/h].
        tau = prognosis ./ l; %Converts the harvest to time [h].
end

%Creating a counter that increases until a day has passed.
KMax = 0; counter = 0;
while KMax < 10
    counter = counter + 1;
    KMax = KMax + tau(counter);
end
KappaMax = ceil(tau(end)); %The time required to harvest the largest field.
fprintf('The␣maximum␣amount␣of␣visited␣fields␣in␣a␣day␣is:␣%d.\n', counter)
fprintf('The␣largest␣field␣takes␣(rounded␣up)␣%g␣hours␣to␣harvest.\n', KappaMax)
```

B.4 Modellexemplet

Listing 4: Matlab-kod för att lösa modellexemplet grafiskt.

```
%Clears previous runs.
clc; clf; clearvars;

xv = 0:100; %Defines a vector for the amount of wheat bought.
c1 =@(x) x; %Constraint 1: -xv+xm<=0 .
c2 =@(x) -x + 100; %Constraint 2: xv+xm-100<=0.
normal = @(x) 48.5 -x*1/6;

%Plotting the constraints.
hold on; axis equal
plot(xv, c1(xv),'k--', LineWidth=1);
plot(xv, c2(xv),'k--', LineWidth=1);
c3 = xline(20, 'k--', LineWidth=1); %Constraint 3: -xv+20<=0.
c4 = yline(20, 'k--', LineWidth=1); %Constraint 4: -xm+20<=0.

%Filling the feasible region, plotting the vector field and its normal plane.
fill([20 80 50], [20 20 50], 'b', 'FaceAlpha', 0.2, LineWidth=2);
quiver(50, 40, 4, 24, 'r', LineWidth=1.5); %Scaled down by a factor 1000.
plot(xv(30:70), normal(30:70), '--', 'Color', '#800020', LineWidth=1);

xlim([15 85]); ylim([15 85]);
xlabel('Antal␣hektar␣vete␣[ha]'); ylabel('Antal␣hektar␣majs␣[ha]')
title('Grafisk␣illustration␣av␣modellexempel');
```

B.5 CPLEX-implementation av det verkliga exemplet

Listing 5: CPLEX-implementation av det verkliga exemplet

```
//constants
int nF = ...; //number of fields
int nD = ...; //number of days
int nK = ...; //number of orders

//Set
 range F = 1..nF; //fields
 range D = 1..nD; //days
 range R = 1..(nD - 1); //days - 1
 range Dr = 2..nD; //days except first
 range K = 1..nK; //orders
 range Kr = 1..(nK - 1); //orders - 1
 range D_1 = 1..(nD+1); //days +1

 //Variables
 dvar float kappa[F][D];
 dvar float w[F][D_1];
 dvar float s[D];
 dvar float rho[D];
 dvar float r[D];
 dvar boolean f[F][D][K];
 dvar boolean z[F][D][K];
 dvar boolean u[F][F][R];
 dvar boolean x[F][F][D][Kr];
 dvar boolean s_h[D];
 dvar boolean k_h[D];

 //functions of sums for costfunction
 dexpr int g[i in F, p in F] = sum(j in R) u[i][p][j]; //u-term
 dexpr int y[i in F, p in F] = sum(j in D, k in Kr) x[i][p][j][k]; //x-term
```

```
 //functions of sums for constraints regarding time
 dexpr int v[i in F, j in D] = sum(k in K) f[i][j][k]; //f-term
 dexpr int n[i in F, p in F, j in D] = sum(k in Kr) x[i][p][j][k]; //x-term

 //Independent parameters
 float c = ...; //cost of transporting loader per hour
 float h = ...; //speed loader
 float l = ...; //capacity loader
 float alpha = ...; //penalty
 float beta = ...; //penalty

float q[D] =...; //quota per day
float phi[F]; //quota fields
float tau[F]; //time to load fields
float t[F][F]; //timematrix


execute { //Extract harvest for every field
  var i = 1;
  var f = new IloOplInputFile("field_1.csv");

  while (!f.eof) {
    var data = f.readline().split(";");
    if(i <=10){
    phi[i] = Opl.floatValue(data[1]);
    tau[i] = phi[i]/l;} //loadertime
    i = i + 1;
  }
}


execute { //Extract time to drive between fields
  var f = new IloOplInputFile("time_1.csv");
  var i = 1;
  while (!f.eof) {
    var data = f.readline().split(",");
    var j = 0;
    while (j < 10){
        if (data[j] != null){
                if(i <=10){
                        t[i][j+1] = data[j];
                                        }
                                                        }
                j = j + 1;
                                }
        i = i + 1;
  }
}

// Big-M parameters
 int M_hat = 15; //time
 float M_w = 0.245; //rest
 int M_max = 1500; // larger than largest tau[i]


//costfunction
minimize c*sum(i in F, p in F) t[i][p]*(g[i][p] + y[i][p])
                + sum(j in D) (alpha*rho[j] + beta*r[j]);
//Constraints
subject to{

   //Maximum one field per day and order
   forall(j in D, k in K)
         ct1: sum(i in F) f[i][j][k] <= 1;

   //Maximum one order per fields and day
   forall(i in F, j in D)
     ct2: sum(k in K) f[i][j][k] <= 1;

   //Time day one
   ct3: sum(i in F) tau[i]*v[i][1] + sum(i in F, p in F) t[i][p]*n[i][p][1]
        - (sum(i in F) kappa[i][1]) + s[1] == 10;

   //Time day j
   forall(j in Dr)
     ct4: sum(i in F) tau[i]*v[i][j] + sum(i in F, p in F) t[i][p]*n[i][p][j]
     + sum(i in F) (kappa[i][(j-1)] - kappa[i][j])
     - (sum(i in F) tau[i]*u[i][i][(j-1)]) + s[j]== 10;

   //Quota day 1
   ct5: sum(i in F) phi[i]*v[i][1] - l*(sum(i in F) kappa[i][1]) == q[1] + rho[1] - r[1];

   //Quota day j
   forall(j in Dr)
```

```
  ct6: sum(i in F) phi[i]*v[i][j] + l*(sum(i in F) (kappa[i][(j-1)] - kappa[i][j]))
  - l*(sum(i in F) tau[i]*u[i][i][(j-1)]) == q[j] + rho[j] - r[j];

//The order is one next day if there exist a rest on the field
forall(i in F, j in R)
  ct7: f[i][(j+1)][1] >= kappa[i,j]/M_max - M_w/(M_max);

//Constraint definition of z
forall(i in F, j in D, k in Kr)
  ct8: z[i][j][k] >= f[i][j][k] - sum(p in F) f[p][j][(k+1)];

//Constraint definition of z
forall(i in F, j in D)
  ct9: kappa[i][j] <= M_max*sum(k in Kr) z[i][j][k];

//Correct order
forall(j in D, k in Kr)
  ct10: sum(p in F) f[p][j][(k+1)] <= sum(i in F)f[i][j][k];

//Constraint definition av of x
forall(i in F, p in F, j in D, k in Kr)
  ct11: x[i][p][j][k] >= 0.5*(f[i][j][(k)] + f[p][j][(k+1)] - 1);

//Number of moves
ct12: sum(i in F, p in F, j in D, k in Kr) x[i][p][j][k] + sum(i in F, p in F, j in R) u[i][p][j]
      - sum(i in F, j in R) u[i][i][j] == nF - 1;

//Constraint definition of u
forall(i in F, p in F, j in R)
  ct13: u[i][p][j] >= 0.5*(f[p][(j+1)][1] - 1 + sum(k in K) z[i][j][k]);

//One field that is last per day
forall(j in D)
  ct14: sum(i in F, k in K) z[i][j][k] == 1;

//Number of visits per field
forall(i in F)
  ct15: sum(j in D, k in K) f[i,j,k] == 1 + sum(j in R) u[i][i][j];

//Constraint definitionen s_h
forall(j in D)
  ct16: s_h[j] <= M_hat*s[j];

//Constraint definitionen s_h
forall(j in D)
  ct17: s_h[j] >= s[j]/M_hat;

//Constraint definitionen k_h
forall(j in D)
  ct18: k_h[j] <= M_max*sum(i in F) kappa[i][j];

//Constraint definitionen k_h
forall(j in D)
  ct19: k_h[j] >= sum(i in F) kappa[i][j]/M_max;

//Either rest or time left
forall(j in D)
  ct20: k_h[j] + s_h[j] <= 1;

//Rest non negative
forall(i in F, j in D)
  ct21: kappa[i][j] >= 0;

//Time left non negative
forall(j in D)
  ct22: s[j] >= 0;

//Not enough loaded non negative
forall(j in D)
  ct23: rho[j] >= 0;

//Too much loaded non negative
forall(j in D)
  ct24: r[j] >= 0;

//Amount left first day
forall(i in F)
  ct25: w[i][1] == l*tau[i];

//Amount left after last day
forall(i in F)
  ct26: w[i][(nD+1)] <= M_w*l;

//Rest left on last day
```

```
  forall(i in F)
    ct27: kappa[i][nD] <= M_w;

  //Con not load if no sugarbeets left
  forall(i in F, j in D)
    ct28: sum(k in K) f[i][j][k] <= w[i][j]/(M_w*l);

  //Amount loaded day one
  forall(i in F)
    ct29: w[i][1] - w[i][2] == l*(tau[i]*sum(k in K) (f[i][1][k]) - kappa[i][1]);

  //Amount loaded day j
  forall(i in F, j in Dr)
    ct30: w[i][j] - w[i][j+1] == l*(tau[i]*sum(k in K) (f[i][j][k]) +
                                          (kappa[i][j-1] - kappa[i][j]) - tau[i]*u[i][i][j-1]);

  //Balance amount
  forall(i in F, j in D)
    ct31: w[i][j] >= w[i][(j+1)];

  //Con not move to same field
  forall(i in F, j in D, k in Kr)
    ct32: x[i][i][j][k] == 0;

  //Starting point
  ct33: f[1][1][1] == 1;
};
```

B.6 Indata till det verkliga exemplet i CPLEX

Listing 6: CPLEX-implementation

```
nF = 10; //number of fields
nD = 5; //number of days
nK = 5; //number of orders

//Independent parameters
c = 1300;
h = 20;
l = 163;
alpha = 10;
beta = 10;

q = [909, 909, 909, 909, 909]; //quota per day
```