CHALMERS | GÖTEBORGS UNIVERSITET

Modellering och optimering av schemaläggning vid en ridskola

En fallstudie i heltalsprogrammering

Kandidatarbete inom civilingenjörsutbildningen vid Chalmers

Rasmus Einarsson
Patrik Johansson
Oskar Redlund
Joakim Widstrand

Institutionen för matematiska vetenskaper
Chalmers tekniska högskola
Göteborgs universitet
Göteborg 2010

Modellering och optimering av schemaläggning vid en ridskola

En fallstudie i heltalsprogrammering

Kandidatarbete i matematik inom civilingenjörsprogrammet Automation och mekatronik vid Chalmers
Oskar Redlund

Kandidatarbete i matematik inom civilingenjörsprogrammet Maskinteknik vid Chalmers
Joakim Widstrand

Kandidatarbete i matematik inom civilingenjörsprogrammet Teknisk matematik vid Chalmers
Patrik Johansson

Kandidatarbete i matematik inom civilingenjörsprogrammet Kemiteknik med fysik vid Chalmers
Rasmus Einarsson

Handledare: Michael Patriksson
Ann-Brith Strömberg

Examinator: Carl-Henrik Rune Fant

Institutionen för matematiska vetenskaper
Chalmers tekniska högskola
Göteborgs universitet
Göteborg 2010

Sammanfattning

Syftet med detta arbete är att analysera möjligheterna att använda sig av matematisk modellering och optimering för att förbättra schemaläggningen vid en ridskola. Projektet har genomförts i samarbete med Billdals ridklubb, där schemat idag läggs manuellt, vilket är tidskrävande. I rapporten presenteras en linjär heltalsmodell vilken använts för att optimera ridskolans schema med hjälp av optimeringslösaren CPLEX. En heuristisk algoritm har utvecklats för att möjliggöra jämförelse av lösningar och lösningstider mellan lösningar från den linjära heltalsmetoden och den heuristiska metoden. De två metoderna har visat sig ha olika fördelar. Resultaten i rapporten visar att ridskolans schema undantagsvis bryter mot de kriterier som formulerats i modellen, men liknar lösningarna från den linjära heltalsoptimeringen. Vi bedömer att matematisk optimering med datorverktyg kan vara till stor hjälp vid schemaläggning på ridskolor.

Abstract

The purpose of this work is to analyze the possibilities of using mathematical modeling and optimization to improve the schedule for a horseback riding school. The work has been done in collaboration with a riding school in Billdal, Sweden, where scheduling is presently done manually, which requires time and effort. A linear integer optimization model has been developed and solved with the optimization software CPLEX. In addition, a heuristic algorithm has been developed for comparison of solutions and computation times between the linear integer method and the heuristic method. The two methods have shown different strengths. Results show that the existing schedule at the riding school ignores some of the restrictions present in the model. We conclude that mathematical optimization can be a very valuable tool in scheduling at horseback riding schools.

Innehåll

Förord: Bidragsrapport

Under projektets gång har en grupploggbok samt en individuell tidslogg förts, dessa ligger till grund för beskrivningen av vad varje projektmedlem bidragit med. I denna sektion presenteras hur arbetsfördelning inom gruppen har fungerat. Nedan presenteras också författarreferenser för rapportinnehållet.

Grupparbete

Medlemmarna i projektgruppen har inte tilldelats några specifika ansvarsområden eftersom en totalöverblick av projektet har bedömts vara viktig för att effektivt bidra till arbetet. Gruppen haft veckovisa möten med få undantag. På dessa möten har den aktuella statusen av projektet diskuterats samt att planer och tilldelning av arbetsuppgifter gjorts inför kommande vecka. Gruppen har syftat till att försöka hålla så kreativa möten som möjligt där detaljer angående projektet och rapporten diskuterats. Även om det mesta faktiska arbetet utförts utanför gruppmötena har de flesta idéer och förslag uppkommit på just dessa. Gruppen har även använt sig av kontakt via e-post mellan mötena där idéer och frågor behandlats kontinuerligt.

Huvudsakligen har den linjära heltalsmodellen konstruerats av Rasmus. Den heuristiska algoritmen har huvudsakligen utvecklats av Patrik. Generering av indata till modellen samt testkörningar och analyser har huvudsakligen utförts av Oskar och Joakim. Det bör dock poängteras att samtliga inblandade har tagit ansvar och bidragit med arbete och diskussion inom samtliga arbetsområden.

Rapport

Här presenteras de huvudsakliga författarna av de olika rapportavsnitten. Författare inom parentes indikerar ett i förhållande betydligt mindre bidrag. Samtliga författare har dock bidragit med förslag till alla avsnitt.

Inledning: Joakim / Oskar

Teori: Patrik

Problembeskrivning: Rasmus / (Oskar)

Metod: Rasmus / (Patrik)

Resultat: Patrik / Oskar / Joakim

Diskussion: Samtliga författare

Slutsats: Samtliga författare

Förkortingslista

AMPL - A Mathematical Programming Language
BB - Branch and Bound
IP - Heltalsprogrammering (*integer programming*)
LP - Linjärprogrammering
MIP - Blandad heltalsprogrammering (*mixed integer programming*)
MPS - Mathematical programming system

1 Inledning

Schemaläggningsproblem återfinns inom många områden i samhället, allt från stora och komplicerade problem till mindre, enklare planering. Under flera decennier har schemaläggning betraktats som ett matematiskt optimeringsproblem, och en stor mängd material har publicerats inom området. De första datorbaserade optimeringsberäkningarna kom till i slutet av andra världskriget när Storbritannien samlade forskare för att effektivisera strategi och taktik [1]. Efter kriget tog utvecklingen mera fart och sedan dess har matematisk optimering blivit en central del av så kallad operationsanalys, det vill säga formell analys för strategiskt beslutsfattande. George Dantzig, som arbetade för amerikanska flygvapnet under andra världskriget och senare formulerade simplexmetoden (se teoriavsnittet 2.3), har vid två tillfällen [2, 3] beskrivit denna bakgrund i större detalj.

Optimering kan beskrivas som att på ett så effektivt sätt som möjligt använda begränsade resurser för att möta vissa behov. Det handlar om att maximera eller minimera någon målfunktion. Exempelvis kan målet vara att maximera en vinst, maximera antalet produkter som kan fraktas och så vidare. Exempel på minimeringsproblem är minimering av kostnader, minimering av antalet anställda eller minimering av körda kilometer. Williams [4] tar upp klassiska och tydliga exempel på när optimering används. Han behandlar problem som uppstår vid fabriksplanering, personalplanering, ruttplanering, tillverkningsplanering och dylikt.

Större scheman där många villkor skall uppfyllas blir ofta beräkningsmässigt komplicerade. Lösningstiden för omfattande schemaläggning växer snabbt med problemets storlek, vilket gör avvägningen mellan generalitet och funktionalitet svår [4]. En generellt formulerad modell är i regel mer svårlöst, men har större flexibilitet. Ett sätt att minska lösningstiden är att först dela upp ett större problem i mindre delproblem, som sedan löses separat i sekvens. Enligt Borndörfer, Löbel och Weider [5] kräver omfattande problem stor datorkapacitet.

I litteraturen återfinns många exempel på olika sätt att formulera och lösa schemaläggningsproblem. Burke och Petrovic [6] noterar att sådana problem ofta har lösts framgångsrikt, men att inga generella slutsatser finns gällande vilken metod som lämpar sig bäst i olika situationer. För en överblick över olika typer av schemaproblem och lösningsmetoder hänvisas till Schaerf [7] som går igenom både deterministiska och stokastiska algoritmer, inklusive heuristiska sådana (baserade på mänsklig analys av problemet).

1.1 Bakgrund: schemaläggning på ridskolan i Billdal

Billdals Ridklubb driver en ridskola med drygt 600 elever, alltifrån totala nybörjare till erfarna ryttare som tar hopp- eller dressyrlektioner. Omkring 40 hästar och 10 lärare ska tillsammans täcka de skilda behov som dessa elevgrupper har. Under sommarhalvåret kan mycket av undervisningen ske utomhus, men i praktiken måste ridskolans schema konstrueras så att alla lektioner får plats inne i ridhuset när vädret inte inbjuder till utomhusaktivitet.

Schemaläggningsproblemet vid ridklubben kan sägas bestå av flera delar: att bestämma vilka lektioner som ska hållas vid olika tider, vilka lektionstider eleverna ska tilldelas, vilka hästar som ska användas vid dessa lektioner och vilka lärare som ska leda lektionerna. Dessa val ska göras under vissa begränsningar, till exempel när elever och lärare är tillgängliga, hur många lektioner hästarna orkar per dag och så vidare. Schemats olika delar är kopplade till varandra eftersom vissa val omöjliggör andra. Uppenbara exempel på detta är att människor och djur bara kan vara på en plats samtidigt och att två lektioner inte kan hållas i samma lokal på samma tid. Mera komplicerade samband uppstår exempelvis för att olika hästar lämpar sig för olika typer av lektioner. Om många lektioner av samma typ planeras i följd kan detta resultera i hårda arbetspass för vissa hästar.

I dagsläget görs all schemaläggning på ridskolan manuellt. Elever delas in efter kunskapsnivå i grupper, typiskt med 10–12 elever i varje grupp. Schemat pusslas sedan ihop så att tiderna passar för både elever och lärare. För hästarnas del går det att göra ett basschema, men detta måste modifieras från vecka till vecka eftersom hästar kan vara ur drift eller ha nedsatt kapacitet på grund av skada eller sjukdom.

1.2 Syfte och avgränsningar

Syftet med detta projekt är att kartlägga, modellera och optimera schemaläggningen på Billdals ridklubb. Modellen ska vara så generell att den kan tillämpas på liknande schemaläggningsproblem vid andra ridklubbar. Resultat av optimeringen ska ligga till grund för en bedömning av de praktiska möjligheterna att ta hjälp av datorbaserade optimeringsmetoder vid ridklubbar. Vissa slutsatser gällande schemaläggningen i dagsläget bör också kunna dras utifrån de optimeringsexperiment som kan göras med modellen.

Avsikten är inte att tillverka en färdig schemalösning för detta specifika problem, inte heller att den fullständiga modellen skall vara så generell att den är direkt applicerbar på andra schemaläggningsproblem. Kriterierna för schemat skall formuleras linjärt och målfunktionerna skall vara linjära eller kvadratiska.

1.3 Metod

För att förstå problemet på ridskolan och samla in nödvändiga data utfördes studiebesök med intervjuer, och kontakt hölls sedan med personal vid ridskolan via e-post. En detaljerad lista över de kriterier som finns för schemat bekräftades av personalen och låg sedan till grund för modellen som använts.

Litteraturstudier om matematisk modellering av schemaläggningsproblem gjordes parallellt för att få en bredare uppfattning om ämnet. Vidare utfördes litteraturstudier för att inhämta nödvändig teoretisk kunskap inom matematisk modellerings- och optimeringslära (avsnitt 2).

En linjär modell har byggts för att motsvara de kriterier som specificeras i problembeskrivningen (se avsnitt 3). Programvarorna AMPL och CPLEX har använts för att lösa optimeringsproblemet. En närmare beskrivning av dessa programvaror och bakomliggande optimeringsteori återfinns i teoriavsnittet (se avsnitt 2). Dessutom har en heuristisk lösningsmetod tagits fram för jämförelse med lösningar från AMPL och CPLEX.

1.4 Sammanfattning av rapportens innehåll

Efter denna inledning följer ett teoriavsnitt som introducerar viktiga begrepp inom optimering, särskilt linjär programmering och heltalsprogrammering. Stora delar av rapporten går att förstå utan att vara bekant med den teori som presenteras nedan och i de fall där den teoretiska bakgrunden är viktig hänvisas till lämpliga teoriavsnitt.

I avsnitt 3 redogör vi mer utförligt för schemaläggningsproblemet som introduceras ovan, och hur ett optimeringsproblem kan definieras med hjälp av ett kvalitetsmått för schemat. Avsnitt 4 ägnas åt en beskrivning av hur schemaläggningsproblemet formulerats som en uppsättning linjära ekvationer och olikheter och heltalskrav på variabler, samt hur det linjära problemet sedan lösts med datorverktyg. En heuristisk lösningsmetod för problemet presenteras också i avsnitt 4.

Resultaten av ett antal testkörningar presenteras i avsnitt 5. Utgående bland annat från dessa resultat följer en allmän diskussion om möjliga lösningsmetoder för detta och andra schemaläggningsproblem i avsnitt 6.

2 Teori

I detta teoriavsnitt beskrivs vanligen förekommande termer inom optimeringslära. Avsnittet börjar med en kort introduktion till matematisk modellering och nyttan av denna. Sedan följer en beskrivning av varför konvexa och linjära problem är särskilt intressanta. En framstående metod för att lösa problem av denna typ presenteras efter detta. Ett avsnitt ägnas därefter åt heltalsprogrammering och lösningsmetoder för sådana modeller. Till sist beskrivs heuristiska metoder och använda datorverktyg för matematisk programmering.

2.1 Modellering och matematisk programmering

Inom vetenskap förekommer modeller av alla möjliga slag. En modell i matematisk bemärkelse är en förenklad beskrivning av ett system eller en företeelse. Ett forskningsområde där matematisk modellering är en av grundstenarna är så kallad operationsanalys. Inom detta område används matematiska modeller till stöd vid strategiskt beslutsfattande. Dessa modeller antar en relativt abstrakt form för att beskriva verkligheten på ett ändamålsenligt vis. Tanken med operationsanalys är att en modell bestående av matematiska relationer som till exempel ekvationer, olikheter och logiska samband skall ha motsvarigheter i verkligheten, som till exempel fysiska begränsningar och konsekvenser av beslut [4].

Enligt Williams [4] fyller en matematisk modell oftast något eller några av följande syften:

(i) det faktiska byggandet av modellen avslöjar samband vilka annars inte hade varit uppenbara;

(ii) möjligheten att analysera ett problem matematiskt kan ge förslag till beslut och handling som annars inte upptäckts;

(iii) att frikoppla något från verkligheten ger möjligheten att kunna experimentera med det utan allvarliga konsekvenser, eller allt för höga kostnader.

Inom operationsanalys används modeller för att optimera exempelvis utnyttjandet av en resurs. Kvalitetsmåttet på nyttjandet, det vill säga det som skall maximeras eller minimeras, kallas modellens målfunktion. De förutsättningar under vilka målfunktionen är giltig kallas bivillkor. Bivillkoren är en uppsättning begränsningar som tillsammans beskriver ett område inom vilket lösningen till problemet tillåts ligga. Detta område kallas problemets *tillåtna område*. För att formulera och lösa en modell används matematisk programmering. Williams betonar att när det talas om matematisk programmering menas något annat än programmering i en datormiljö. Matematisk programmering har snarare betydelsen ”programmering” i bemärkelsen ”planering”, och har alltså inte nödvändigtvis något med datorer att göra [4]. Metoder för lösning av matematiska programmeringsproblem är dock anpassade för att lösas med datorer. I nästa avsnitt ges en matematisk förklaring till varför programmeringsproblem med konvexa tillåtna områden är särskilt intressanta.

2.2 Konvexa problem

En *konvex mängd* är en mängd i vilken ett rät linje mellan två godtyckligt valda punkter i mängden helt och hållet ligger inuti mängden. En funktion är *konvex* om området ovanför funktionens graf är en konvex mängd. Med andra ord är en funktion konvex om det gäller, för vilka två punkter x_1 och x_2 som helst i funktionens definitionsmängd samt för alla $t \in (0, 1)$, att

$$f(tx_1 + (1-t)x_2) \leq tf(x_1) + (1-t)f(x_2).$$

Funktionen är istället *konkav* om den omvända olikheten gäller.

Ett matematiskt programmeringsproblem kallas konvext om det avser att minimera en konvex funktion över en konvex mängd [4]. Problemet är också konvext om det avser att maximera av en konkav funktion över en konvex mängd. En fundamental egenskap hos konvexa problem är att om en optimal lösning existerar så är denna en globalt optimal lösning. För ett konvext problem innebär denna egenskap att om det visas att en lösning är ett lokalt optimum, så är lösningen också ett globalt optimum.

Ett intressant område inom optimering består av de modeller vars målfunktion och bivillkor kan beskrivas av linjära ekvationer och olikheter [4]. Utifrån definitioner inses att problem som beskrivs av linjära begränsningar och en linjär målfunktion är konvexa. Problem av denna typ modelleras med så kallad linjärprogrammering (LP, *linear programming*). Linjära problem har tillägnats särskilt mycket intresse historiskt jämfört med icke-linjära problem eftersom linjära problem i regel är lättare att lösa [4].

Det finns många modelleringsproblem för vilka bivillkoren kan uttryckas linjärt men den resurs som skall optimeras bättre beskrivs med ett icke-linjärt uttryck. Det finns till exempel många fall då en kvadratisk målfunktion är bättre lämpad än en linjär approximation [4]. Problem med kvadratisk målfunktion och linjära bivillkor kallas kvadratiska programmeringsproblem (QP, *quadratic programming*). Många av de metoder som används för att lösa linjära problem kan anpassas till att lösa konvexa problem med en konvex kvadratisk målfunktion [4].

2.3 Linjärprogrammering och simplexmetoden

Av alla tänkbara optimeringsproblem är många svåra eller omöjliga att beskriva med linjära modeller. Om det är möjligt att beskriva ett verkligt problem linjärt utan att göra allt för grova approximationer finns dock potentiellt mycket att vinna enligt Williams [4]. Fördelen är att problemets struktur då kan utnyttjas för att lösa det på ett effektivt sätt. Den vanligaste metoden för att lösa linjära programmeringsproblem är den så kallade simplexmetoden [8]. Följande beskrivning av simplexmetoden är baserad på Nash [8] och likheter i såväl tankegång som beteckningar förekommer.

Simplexmetoden

Simplexmetoden utvecklades under 1940-talet då linjärprogrammering blev populärt för att utföra ekonomiska och militära beräkningar. Andra metoder utvecklades också men över tid har de flesta av dessa fått ge vika för simplexmetodens överlägsna effektivitet [8]. Även om problem idag är mycket större och beräkningskraften likaså har simplexmetoden utvecklats och anpassats till dagens behov. Det är inte förrän på senare tid som egentliga alternativ till simplexmetoden har trädit fram [8].

Ett system av linjära begränsningar beskriver en konvex polyeder och därför gäller att det tillåtna området till samtliga linjära programmeringsproblem är konvexa mängder. En viktig egenskap hos konvexa polyedrar är att dess extrempunkter består av polyederns hörnpunkter. Detta innebär att den optimala lösningen till ett linjärt programmeringsproblem, om den existerar, finns bland den tillåtna polyederns hörnpunkter. Att endast betrakta dessa punkter är simplexmetodens grundtanke. Dessutom gäller för konvexa problem att en lokalt optimal lösning också är en globalt optimal lösning. Det räcker alltså att visa att en lösning till ett linjärt problem är lokalt optimal för att den också visats vara globalt optimal.

Ett LP-problem skrivs på standardform

$$\begin{aligned} \text{minimera} \quad & z = c^T x, \\ \text{då} \quad & Ax = b, \\ & x \geq 0, \end{aligned} \tag{1}$$

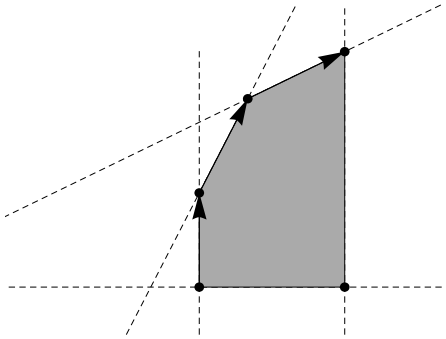
där $b \geq 0$, $c \in \mathbb{R}^n$, $b \in \mathbb{R}^m_+$, $A \in \mathbb{R}^{m \times n}$ och $x \in \mathbb{R}^n$. Här kallas matrisen A för bivillkorsmatris. Det finns regler för att konvertera ett godtyckligt linjärt problem till ett linjärt problem på standardform. Ett rimligt antagande är att matrisen A har full radrang. I annat fall är problemet antingen feldefinierat eller så beskriver flera villkor samma begränsning. I det senare fallet kan problemet reduceras så att antagandet gäller utan att problemets lösning eller mängden av tillåtna lösningar förändras [8].

Simplexmetoden är en iterativ metod som löser linjära programmeringsproblem på standardform. Då metoden används för att lösa icke-degenererade problem görs förflyttningar mellan hörnpunkter till den tillåtna polyedern. Degenererade problem uppstår då flera villkor i ett problem beskriver samma begränsning.

Om en vektor x uppfyller likheterna i ett LP-problem på standardform kallas denna *baslösning* om det dessutom gäller att de kolonner i bivillkorsmatrisen som multipliceras med

icke-noll-elementen i vektorn x är linjärt oberoende. Punkten x är en *tillåten baslösning* om det dessutom gäller att $x \geq 0$, det vill säga att punkten uppfyller samtliga villkor för problemet. Baslösningar erhålls genom att fixera $n - m$ element i x-vektorn för det ursprungliga problemet vid värdet noll och sedan lösa den resterande delen av ekvationssystemet $Ax = b$. De variabler som löses ut kallas basvariabler och de som satts till noll kallas icke-basvariabler. De tillåtna baslösningarna till ett linjärt problem är extrempunkterna till den tillåtna polyedern. Vektorn x kallas en *optimal tillåten baslösning* om dess målfunktionsvärde är lägre än eller lika med målfunktionsvärdet för alla tillåtna baslösningar till det linjära programmet.

Simplexmetoden arbetar genom att utifrån en given första tillåten baslösning undersöka huruvida denna lösning är optimal. Om den inte är optimal väljer metoden en tillåten riktning längs vilken målfunktionen minskar och en förflyttning görs mot en tillåten baslösning längs denna riktning. Denna procedur upprepas sedan. Simplexmetoden gör förflyttningar från en tillåten baslösning till en närliggande tillåten baslösning genom att ta bort och lägga till en basvariabel i taget. Geometriskt innebär detta att flytta sig längs kanterna till den tillåtna polyedern (se figur 1).



Figur 1: Konceptskiss över simplexmetoden i två dimensioner. Med metoden genomsöks den tillåtna polyederns extrempunkter, det vill säga dess hörnpunkter. De streckade linjerna definierar problemets tillåtna område. Punkterna är de tillåtna baslösningarna.

I det generella fallet kan simplexmetoden beskrivas på följande vis för lösning av ett linjärt problem på standardform (1).

Egenskaperna hos en baslösning, tillsammans antagandet att matrisen A har full radrang, gör det möjligt att dela upp komponenterna x i komponenterna x_N och x_B. Vektorn x_B består av basvariablerna vars koefficienter i bivillkorsmatrisen A motsvarar den inverterbara basmatrisen B. Vektorn x_N består av alla icke-basvariabler, vilka har värdet noll.

Om x är tillåten baslösning med variabler ordnade så att

$$x = \begin{pmatrix} x_B \\ x_N \end{pmatrix},$$

där x_B är en vektor med ingående basvariabler och x_N är den nuvarande (just nu nollvektorn) vektorn av icke basvariabler, kan målfunktionen skrivas som:

$$z = c_B^T x_B + c_N^T x_N,$$

där koefficienterna för basvariablerna finns i c_B och koefficienterna för icke-basvariablerna finns i c_N. Bivillkoren kan då skrivas som

$$Bx_B + Nx_N = b,$$

vilka kan uttryckas som

$$x_B = B^{-1}b - B^{-1}Nx_N. \tag{2}$$

Substitution av x_B i formeln för z ger att

$$z = c_B^T B^{-1} b + (c_N^T - c_B^T B^{-1} N) x_N.$$

Om y definieras som $y = (c_B^T B^{-1})^T = B^{-T} c_B$ kan z uttryckas som

$$z = y^T b + (c_N^T - y^T N) x_N.$$

De aktuella värdena på basvariablerna och målfunktionen fås genom att sätta $x_N = 0$, det vill säga

$$x_B = \hat{b} = B^{-1} b \qquad \text{och} \qquad \hat{z} = c_B^T B^{-1} b.$$

För att underlätta undersökningen av hur målfunktionen påverkas av icke-basvariablerna betecknas $\hat{c}_j$ som elementet i vektorn $\hat{c}_N^T \equiv (c_N^T - c_B^T B^{-1} N)$ motsvarande vektorn x_j. Koefficienten $\hat{c}_j$ kallas den *reducerade kostnaden* för x_j. Målfunktionen kan skrivas som

$$z = \hat{z} + \hat{c}_N^T x_N.$$

Om icke-basvariabeln x_j tillskrivs ett noll-skiljt-värde ϵ kommer målfunktions värde alltså att förändras med $\hat{c}_j \epsilon$.

För att undersöka huruvida en lösning är optimal studeras vad som händer med målfunktionen om värdet på var och en av icke basvariablerna tillåts öka från noll. Om $\hat{c}_j > 0$ kommer målfunktionens värde att öka. Om $\hat{c}_j = 0$ kommer ingen förändring ske och om $\hat{c}_j < 0$ kommer målfunktionens värde att minska. Det vill säga att om $\hat{c}_j < 0$ för något j kan målfunktionens värde förbättras om x_j ökas från noll. Om den nuvarande basen inte är optimal kan en variabel x_t , sådan att $\hat{c}_t < 0$, väljas att ingå i basen. Denna variabel kallas en inkommande variabel.

När en ny inkommande variabel valts måste det undersökas hur mycket denna tillåts öka utan att begränsningarna på problemet bryts. Detta bestämmer huruvida en variabel skall utgå ur basen. Basvariablerna bestäms enligt (2) och förutom x_t har alla element i vektorn x_N värdet noll. Därför är

$$x_B = \hat{b} - \hat{A}_t x_t$$

där $\hat{A}_t = B^{-1} A_t$, A_t är den t:te kolonnen av A och $\hat{b} = B^{-1} b$. Ett element i vektorn x_B kan skrivas som

$$(x_B)_i = \hat{b}_i - \hat{a}_{i,t} x_t$$

där $\hat{a}_{i,t}$ är elementen i $\hat{A}_t$. Om $\hat{a}_{i,t} > 0$ kommer $(x_B)_i$ att minska då den inkommande variabeln x_t ökar och $(x_B)_i$ blir lika med noll då $x_t = \hat{b}_i / \hat{a}_{i,t}$. Om $\hat{a}_{i,t} < 0$ kommer $(x_B)_i$ öka och om $\hat{a}_{i,t} = 0$ är $(x_B)_i$ oförändrad.

Variabeln x_t kan ökas så länge alla övriga variabler förblir icke-negativa, det vill säga tills den når värdet

$$\bar{x}_t = \min_{1 \le i \le m} \left\{ \frac{\hat{b}_i}{\hat{a}_{i,t}} : \hat{a}_{i,t} > 0 \right\}.$$

Det minsta värdet av ovanstående uttryck bestämmer de nya icke-basvariabeln $(x_B)_i$ och därmed också den nya tillåtna baslösningen, med x_t som den nya inkommande basvariabeln. Tilldelningarna

$$x_B \leftarrow x_B - \hat{A}_t \bar{x}_t \qquad \text{och} \qquad \hat{z} \leftarrow \hat{z} + \hat{c}_t \bar{x}_t$$

bestämmer värdet av basvariablerna samt det nya värdet av målfunktionen i den aktuella basen. Variabeln x_t tilldelas värdet $\bar{x}_t$ och de kvarvarande icke-basvariablerna förblir noll. En av basvariablerna får värdet noll och utgår ur basen.

Om $\hat{a}_{i,t} \le 0$ för alla värden på i kommer ingen av basvariablerna minska då x_t ökar från noll. Alltså kan x_t göras godtyckligt stort. Detta betyder att målfunktionens värde kommer minska utan undre gräns då $x_t \to \infty$ vilket innebär att problemet har en obegränsad optimallösning.

Enligt principerna ovan kan en tydlig bild av simplexalgoritmen ges. Genom att starta från en basmatris B motsvarande en tillåten baslösning $x_B = \hat{b} = B^{-1} b \ge 0$ itererar simplexmetoden genom följande steg.

1. *Optimalitetstest*; Beräkna vektorn $y^T = c_B^T B^{-1}$. Beräkna koefficienterna $\hat{c}_N^T = c_N^T - y^T N$. Om $\hat{c}_N^T \ge 0$ är den nuvarande basen optimal. Annars, välj en variabel x_t sådan att $\hat{c}_t < 0$ som en ny inkommande basvariabel.

2. *Stegtagande*; Beräkna $\hat{A}_t = B^{-1}A_t$, bivillkorskoefficienterna motsvarande den nya basvariabeln. Hitta ett index s som uppfyller

$$\frac{\hat{b}_s}{\hat{a}_{s,t}} = \min_{1 \leq i \leq m} \left\{ \frac{\hat{b}_i}{\hat{a}_{i,t}} : \hat{a}_{i,t} > 0 \right\}.$$

   Variabel x_S skall lämna basen och pivot-elementet är $\hat{a}_{s,t}$. Om $\hat{a}_{i,t} \leq 0$ för alla i är problemet obegränsat.

3. *Uppdatering*; Uppdatera basmatrisen B och vektorn med basvariabler x_B.

Genom att upprepa ovanstående iterationssteg erhålls den lokalt optimala lösningen, alltså den globalt optimala lösningen.
Simplexmetoden är en effektiv metod för att lösa linjära program och används av de flesta moderna optimeringsalgoritmer för lösning av linjära problem [8]. Metoden ställer dock krav på att det undersökta problemet är linjärt vilket inte alltid är fallet. Vid modellering av vissa typer av problem krävs icke-kontinuerliga variabler. Det finns en rad sätt att hantera dessa variabler. Nedan följer en beskrivning av några metoder som ofta kompletterar simplexmetoden vid lösning av denna typ av problem.

2.4 Heltalsprogrammering och branch-and-bound-algoritmen

Heltalsprogrammering (IP, *integer programming*) behandlar problem bestående av heltalsvariabler. En modell som innehåller en blandning av kontinuerliga variabler och heltalsvariabler behandlas av *blandad heltalsprogrammering* (MIP, *mixed integer programming*). Vanligtvis är problem med heltalsvariabler svårare att lösa än linjära problem av samma storleksordning [4]. Inom operationsanalys är heltalsvariabler ofta [4] begränsade till att anta värdena 0 eller 1. Detta kan till exempel motsvara ett ja-eller-nej-beslut. Metoder för att lösa generella heltalsproblem anpassas utan större svårigheter till att lösa rena eller blandade 0–1-problem [9]. Det finns en uppsjö av sätt att lösa MIP-problem men den metod som används mest är den så kallade *branch-and-bound*-metoden [9], som initieras med att att heltalskraven på problemet släpps och problemet löses som ett LP-problem. Detta förfarande kallas LP-relaxering och utnyttjar det faktum att LP-problem vanligen är lättare att lösa än MIP-problem. Om den optimala lösningen av LP-relaxeringen av ett MIP-problem är en heltalslösning är detta också den optimala lösningen till heltalsproblemet. I Williams [9] beskrivs hur branch-and-bound-algoritmen används för att lösa ett typiskt IP-problem. Under nästa rubrik finns en generalisering av denna beskrivning som illustrerar teorin bakom branch-and-bound-algoritmen.

Branch-and-bound-algoritmen

Branch-and-bound-metoden (BB, "förgrena och avgränsa") löser först LP-relaxeringen av ett IP-problem för att sedan återinföra heltalskraven på variablerna steg för steg. För ett IP med två variabler

$$\begin{aligned} \text{minimera} \quad & z = c^T x, \\ \text{då} \quad & Ax = b, \\ & x \geq 0, \end{aligned}$$

där $b \geq 0$, $c \in \mathbb{R}^2$, $b \in \mathbb{R}^2_+$, $A \in \mathbb{R}^{2\times 2}$, $x \in \mathbb{Z}$ och $x = (x_1, x_2)$, kan BB-metoden se ut på följande sätt.

En nod är en instans av problemet, nämligen LP-relaxeringen av IP-problemet, kompletterat med extra begränsningar som successivt införts i överordnade noder. Dessa begränsningar beskrivs nedan. I startnoden löses LP-relaxeringen av IP-problemet. Detta ger lösningen

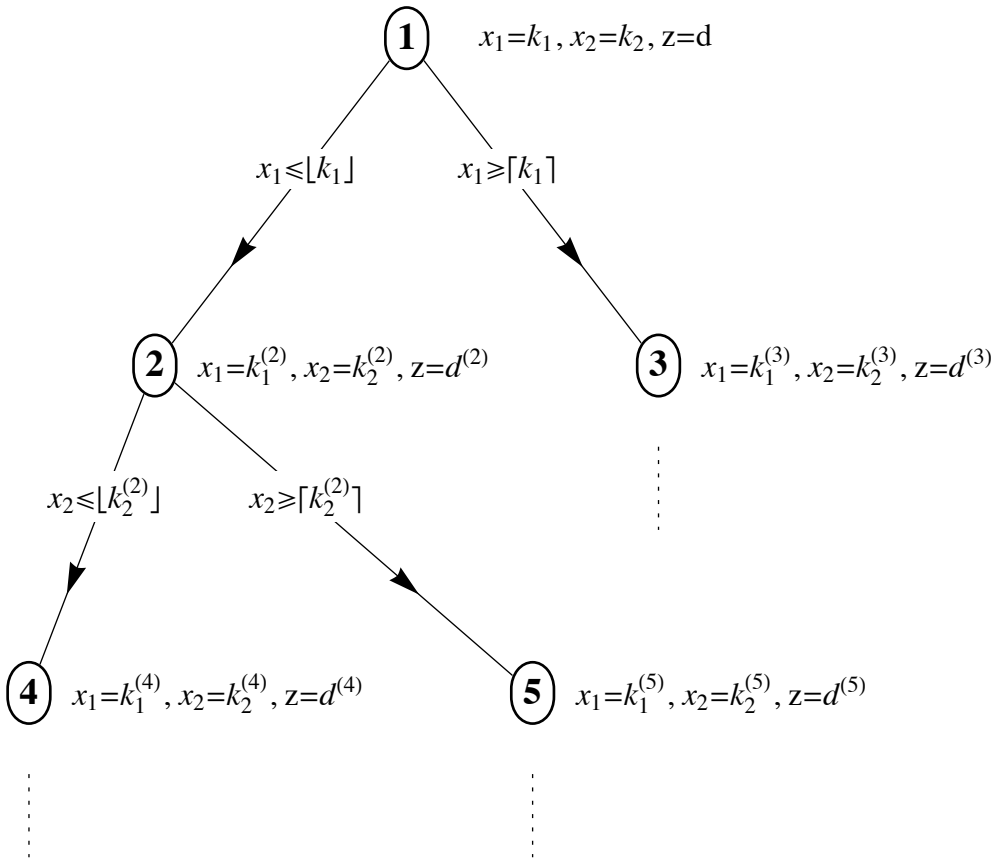
$$x_1 = k_1, x_2 = k_2, z = d.$$

Sedan väljs godtyckligt en variabel vars värde inte är ett heltal och utan att tappa generalitet införs nya villkor på denna variabel. Om x_1 valts införs villkoren

$$x_1 \leq N \qquad \text{och} \qquad x_1 \geq N + 1, \tag{3}$$

i vardera av två nya noder. Observera att dessa villkor inte utesluter några tillåtna värden eftersom värdena mellan N och $N+1$ alla är icke heltal. Däremot utesluts den nuvarande lösningen genom att välja $N = \lfloor k_1 \rfloor$. Införandet av villkoren (3) kan ses som en förgrening i ett träd av delmodeller där olika villkor gäller i de olika grenarna.

Efter denna förgrening väljs en av de underordnade noderna och processen upprepas från denna nod. Det vill säga att LP-relaxeringen av orginalproblemet plus de extra villkor som gäller i denna nod löses och en förgreningsvariabel väljs bland de variabler som har icke-heltaliga värden. Två grenar skapas där denna lösning utesluts med hjälp av villkor på formen (3). Detta ger upphov till ett träd likt det som illustreras i figur 2.



Figur 2: Schematisk bild över ett branch-and-bound-träd. Noderna är numrerade 1–5. De extra villkor som successivt införs vid förgrening står skrivna vid pilarna och är villkor på formen (3). Observera att $N = \lfloor k_1 \rfloor$ i de införda villkoren.

Märk väl att det finns en rad val att göra i denna process. Valet av förgreningsvariabel, ordningen i vilken LP-relaxeringarna i noderna löses samt vilken nod som undersöks efter att en förgrening gjorts kan väljas slumpmässigt. Alternativet är att använda någon regel för att göra dessa val, i hopp om att nå en bra lösning snabbare.

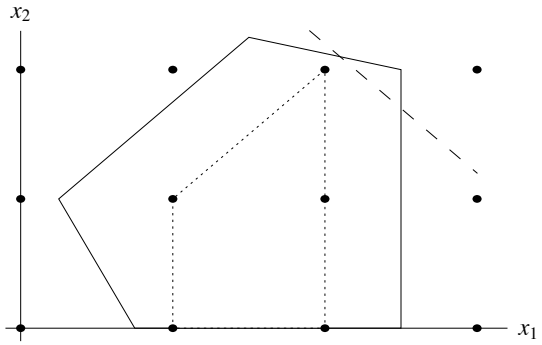
Allt eftersom noderna utvecklas kan en rad saker inträffa. Till exempel kan införandet av ett nytt villkor göra att instansen saknar lösning. Grenen behöver då inte utvecklas vidare. Noden sägs då vara uttömd. Då en heltalslösning erhålls i en nod finns det ingen anledning att undersöka den grenen vidare. Detta eftersom införandet av ytterligare villkor aldrig kan förbättra målfunktionens värde. Värdet på den bästa funna heltalslösningen vid en viss tidpunkt fungerar som en gräns för alla andra noder. Om värdet av målfunktionen av LP-

relaxeringen i någon nod är sämre än detta värde, är det inte nödvändigt att utveckla denna vidare. Den sägs då vara uttömd. Skillnaden mellan en heltalslösning och motsvarande LP-relaxerings lösning kallas *heltalsgap* och kan användas som ett mått på hur nära optimal en heltalslösning är.

Då alla grenar följts till sitt slut, det vill säga då de nått en heltalslösning eller konstaterats vara ointressanta, har den optimala lösningen hittats. Det finns dock inga garantier för att ett träd skall vara uttömligt. Det vill säga att det finns fall då ett oändligt antal noder kan genomsökas utan att en giltig lösning hittas. Detta inträffar dock endast om det tillåtna området för problemet är obegränsat.

Notera att ett infört villkor gäller i alla noder under noden där det införts i trädet. Om branch-and-bound-algoritmen används på ett IP-problem där värdena på variablerna begränsas till att anta 0 eller 1 blir trädet enklare. Varje gren fixerar då värdet av en variabel i alla noder i den gren som definieras av den nod där villkoret införts.

Lösningsrummet till ett linjärt problem i två dimensioner kan illustreras som i figur 3. Detta kan till exempel vara LP-relaxeringen av ett IP-problem. De tillåtna heltalslösningarna är de heltalspunkter som ligger inne i polyedern. Det minsta konvexa område som innesluter dessa punkter kallas problemets *konvexa hölje* av tillåtna heltalslösningar. Detta syns som de prickade linjerna i figur 3. I två dimensioner är gränserna till området räta linjer och i högre dimensioner är de hyperplan. Gränserna till det konvexa höljet kallas *fasetter* och de villkor som ger upphov till dem kallas fasettdefinierande begränsningar. Dessa kan ibland vara svåra att beräkna men ibland kan de approximeras av begränsningar som skär bort en optimal lösning till LP-relaxeringen, men som inte är fasetter [9]. Dessa begränsningar kallas skärande plan (*cutting planes*).



Figur 3: Det konvexa höljet (prickat) av lösningsrummet till LP-relaxeringen av ett IP-problem (heldraget) i två dimensioner, samt ett skärande plan till LP-relaxeringen (streckat). Punkterna markerar heltalslösningar.

Att använda denna typ av bivillkor i kombination med BB-metoden kan förväntas reducera antalet noder att genomsöka. Ett vanligt förekommande problem är dock att det ofta finns väldigt många fasetter till ett IP-problem. I praktiken kan dessa begränsningar (eller andra skärande plan) endast användas iterativt för att utesluta nuvarande icke-heltalslösningar [9]. Detta tillvägagångssätt kan kombineras med BB-metoden genom att applicera bivillkoren på instanserna av IP-problemet i de olika noderna i trädet. Metoden kallas då branch-and-cut.

2.5 Heuristiska lösningsmetoder

I verkligheten ämnar optimering ofta undersöka hur en rad beslut kan kombineras för att det totala resultatet skall bli så bra som möjligt. Detta leder i många fall, som diskuteras i avsnitt 2.4, till problem med variabler begränsade till värdena 1 eller 0, vilka svarar på en ja-nej fråga gällande ett visst beslut. Ett problem med n stycken oberoende variabler av denna typ har totalt 2^n möjliga utfall. Då ett verkligt problem skall undersökas kan antalet beslut vara precis så många. Detta gör att det ibland är omöjligt att undersöka alla tänkbara lösningar eftersom antalet kandidater helt enkelt är för stort. Branch-and-bound-metoden, som diskuteras i avsnitt 2.4, är ett sätt att undvika att söka igenom alla tänkbara lösningar.

En viktig egenskap hos BB-metoden är att den kan garantera optimalitet av en funnen lösning.

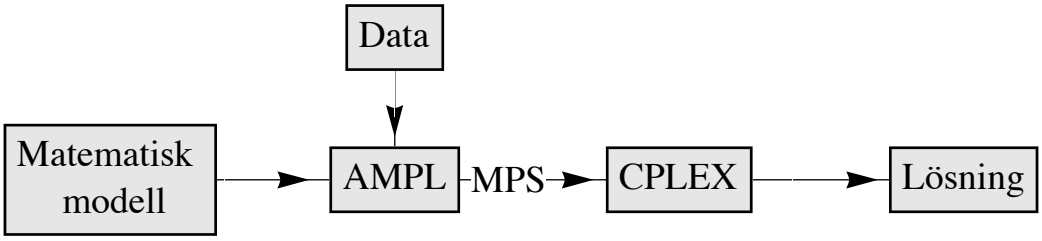
Om det anses ta orimligt lång tid att finna en optimal lösning till ett problem, kan det ibland vara aktuellt att leta efter en suboptimal, men ändå godtagbar lösning. Optimeringsproblem där en suboptimal lösning är tillräcklig kallas semioptimering [10]. Pearl [10] poängterar att det i många verkliga fall är nödvändigt att använda sig av semioptimering. Vidare menar han att en noggrann avvägning måste göras mellan krav på hur nära optimal lösningen skall vara och hur lång lösningstid som är godtagbar. Fördelen med semioptimering är att fokus skiftas från att verifiera optimalitet av en funnen lösning till att plocka fram godtagbara lösningar så effektivt som möjligt. Heuristik är kriterier och metoder som väljer det alternativ, som bedöms vara det effektivaste sättet att nå ett visst mål [10]. *Heuristiska metoder* är metoder som förlitar sig på heuristiska avväganden för att finna en godtagbar lösning.

Heuristiska metoder kan se ut på många sätt; den gemensamma nämnaren är att de bygger på kunskap om problemets struktur. Branch-and-bound-metoden kan till exempel utnyttja heuristiska regler för att finna en bra lösning snabbare. En heuristisk metod för att hitta den kortaste vägen mellan en rad destinationer kan till exempel vara att hela tiden förflytta sig till den obesökta destination som ligger närmast den nuvarande platsen [10]. I detta fall fattas det heuristiska beslutet baserat på kunskap om vilken destination som ligger närmast, samt antagandet om att en kort sträcka i ett närsynt perspektiv förhoppningsvis leder till kortare sträcka totalt. Detta ger sannolikt inte en optimal lösning till problemet, men är oftast bättre än att irra runt planlöst [10]. Det finns inga generella heuristiska metoder för att lösa optimeringsproblem. Metoder måste istället ofta specialkonstrueras för specifika problem.

2.6 Modelleringsverktyg för lösning av linjära problem

Linjära modeller av verkliga problem kan innehålla stora mängder variabler och villkor. Dessa behöver struktureras för att tillåta översättning mellan den människa som formulerat dem och den dator som skall lösa dem. I denna sektion beskrivs hur de programvaror som använts i projektet interagerar. Dessutom ges en kort beskrivning av de olika programvarorna.

Det mest använda formatet för att beskriva LP-problem så att optimeringsverktyg kan tolka dem är det så kallade MPS-formatet (mathematical programming system) [4]. Att formulera ett stort problem i detta format är ofta inte överskådligt och därför används modelleringsprogram för att sköta översättningen mellan matematisk modell och MPS-format. I detta projekt används AMPL för att formulera stora linjära heltalsproblem och optimeringslösaren CPLEX används för att lösa dem. I figur 4 beskrivs kopplingen mellan matematisk modellering och lösning av ett IP-problem.



Figur 4: Strukturen mellan de olika programvarorna som används i detta projekt.

2.6.1 AMPL

Modelleringsspråket AMPL (*A Mathematical Programming Language*) designades och implementerades första gången 1985 [11]. AMPL används för att formulera matematiska modeller i en dator på ett sätt som är intuitivt för användaren. Enligt utvecklarna [11] är en av de största fördelarna med AMPL att uttrycken som språket hanterar i hög grad liknar de vanliga algebraiska motsvarigheterna. AMPL innehåller inte några matematiska optimeringsmetoder. Verktyget består av ett gränssnitt som förmedlar kontakten mellan den matematiska modell

som användaren formulerar, de data som definieras av en probleminstans och den lösningsmetod som används. Den intuitiva arbetsgången med AMPL beskrivs enligt följande procedur: [11]

(i) formulering av den matematiska modellen i generell form;

(ii) inhämtning av data som representerar en viss instans av problemet;

(iii) generering av målfunktion och bivillkor i MPS-format från modellen och data;

(iv) lösning av problemet genom anrop av en extern lösningsprogramvara;

(v) presentation av resultatet och analys av användaren;

(vi) eventuell förfining av problemet och upprepning av proceduren.

2.6.2 CPLEX

CPLEX är ett verktyg för att lösa problem inom matematisk programmering. Version 1.0 släpptes av ILOG 1988 och sedan dess har programmet genomgått betydande utveckling [12]. I detta arbete används CPLEX för att lösa problem inom linjär heltalsoptimering men kan också bland annat också lösa vissa problem inom kvadratisk programmering. CPLEX har sitt ursprung i programmeringsspråket C men är idag också delvis uppbyggt av C++, Java, .NET och Python [13].

För att lösa de mest grundläggande problemen inom linjär programmering, där variablerna i målfunktionen är kontinuerliga, använder sig CPLEX främst av metoder baserade på simplexalgoritmer [13]. CPLEX är ett omfattande verktyg och erbjuder användaren möjligheten att anpassa lösningsmetoden för att passa det problem som skall lösas. För lösning av MIP används BB-metoden, heuristiker och generering av skärande plan och fasetter [13] (se avsnitt 2.5 och 2.5).

3 Problembeskrivning

I avsnitt 1.1 introducerade vi schemaläggningsproblemet på ridskolan. Kort sagt ska schemat passa både elever och lärare utan att hästarna blir överbelastade. I detta avsnitt presenteras problemet mer i detalj, men först följer några ord om hur schemat läggs idag.

3.1 Schemaläggning på ridskolan

All schemaläggning på ridskolan sker manuellt. De elever som ansöker om plats på ridskolan har möjlighet att ange tider då de är tillgängliga. Eleverna delas in i grupper och veckoschemat läggs så att tiderna ska passa så många som möjligt. Slutligen erbjuds eleverna en lektionstid som de kan tacka ja eller nej till. För hästarnas del går det att göra ett basschema, men detta måste modifieras från vecka till vecka eftersom hästar kan vara ur drift eller nedsatta på grund av skada eller sjukdom.

I dagsläget finns ett basschema för ridskolans elevgrupper och deras lektionstider. När vi hänvisar till ridskolans *befintliga schema* är det detta lektionsschema för elever som avses, dock inte nuvarande tilldelning av hästar och lärare. I det befintliga schemat finns lektionsplats för 605 elever.

I praktiken finns inga strikta regler för hur schemat ska läggas, utan personalen avgör själva vad som är tillåten belastning för hästarna, vilka arbetstider som ska gälla för lärarna, och så vidare. För att behandla problemet med matematiska verktyg är det dock nödvändigt att definiera entydigt vad som är tillåtet och inte. För detta ändamål har en lista med kriterier som ställs på schemat formulerats med hjälp från ridskolans personal. Denna lista speglar dock inte perfekt de överväganden som görs i verkligheten, eftersom en människa med rätt erfarenhet kan göra avvägningar och undantag som är svåra eller omständliga att formulera matematiskt.

3.2 Elever

Eleverna ska bokas in på tider där de angivit att de är tillgängliga. De bildar grupper så att alla elever i varje grupp har likvärdiga kunskaper. Detta löses genom att lektionstillfällen tilldelas en viss *lektionstyp*, exempelvis nybörjare, nivå 1–4 eller dressyr, och för varje elev bestäms alltså vilken lektionstyp som ska bokas. Olika lektionstyper pågår olika länge – exempelvis är det så kallade ridlekis 30-minuterslektioner, hopplektionerna 60 minuter och övriga lektioner är 45 minuter. Lektionstyperna har också olika antal elever. Typerna presenteras i tabell 1.

Tabell 1: Lektionstyperna på Billdals ridskola.

Lektionstyp	Längd [minuter]	Antal elever (min–max)
Ridlekis	30	11–13
Nybörjare	60	10–12
Nivå 1	45	10–12
Nivå 2	45	10–12
Nivå 3	45	8–10
Nivå 4	45	8–10
Hoppning	60	6–8
Dressyr	45	5–7
Vuxna nybörjare	30	8–10
Vuxna nivå 1	45	8–10
Vuxna nivå 2	45	8–10
Vuxna nivå 3	45	8–10
Vuxna nivå 4	45	8–10
Hoppning vuxna	60	6–8

3.3 Hästar

Ridskolans 40 hästar har olika storlek, träningsnivå, personlighet, fysisk kondition och eventuellt ytterligare faktorer som avgör vilka elever de lämpar sig för. För schemaläggningens skull kan dock förhållandet mellan hästar och elever representeras på ett enkelt sätt genom att en lista tas fram för varje häst, som anger vilka lektionstyper som hästen kan delta på. Att en häst tillåts delta vid lektioner av en viss typ innebär alltså även att hästen är lämplig för en typisk elev på denna lektionstyp.

Vidare finns det begränsningar i arbetsbelastning för hästarna. Dels finns ett maximalt antal lektioner *per dag* och häst, dels finns ett tak för antal lektioner *i direkt följd*. Dessa begränsningar anges individuellt för varje häst. Beroende på lektionstyp varierar också antalet lektioner som kan tillåtas – exempelvis kan hästar typiskt gå tre lektioner i rad, men om någon av lektionerna är mer ansträngande (exempelvis hoppning eller nivå 4) tillåts bara två lektioner i rad.

3.4 Lokaler

På ridskolan finns i dagsläget plats för två parallella lektioner i ridhuset. Alltså kan aldrig fler än två lektioner pågå samtidigt.

3.5 Lärare

På ridskolan finns 10 lärare som delar på veckans lektioner. Varje lektion har en lärare. Dessa är tillgängliga på olika tider och får förstås endast bokas in då de är tillgängliga. I likhet med hästarna finns en gräns för hur många lektioner i följd en lärare kan leda. Den totala arbetstiden bestäms genom att ange ett maximalt antal lektioner per vecka och lärare.

3.6 Ytterligare önskemål

Utöver det ovanstående finns flera önskemål gällande schemat, som inte är helt nödvändiga att tillgodose. Vi noterar att den enklaste schemalösningen enligt ovanstående kriterier vore att inte erbjuda några elever plats alls – då skulle ju varken lärare eller hästar krävas! Men sådana triviallösningar är förstås ointressanta. Slutsatsen blir att vi vill maximera antalet elever utan att slita ut personal eller hästar. Andra sekundära mål (optimeringsmål) kan vara att fördela arbetet jämnt mellan hästarna, att göra schemat så kompakt som möjligt för lärarna, eller att skapa ett rättvist schema för lärare eller hästar.

4 Metod

Den metod som primärt använts för att lösa schemaläggningsproblemet är linjär heltalsprogrammering (se avsnitt 2). För detta ändamål har en linjär modell konstruerats för att motsvara de kriterier som specificeras i problembeskrivningen ovan (avsnitt 3).

Optimeringsproblemet har implementerats i AMPL och lösts med CPLEX (se avsnitt 2.6.1). Dessutom har en heuristisk lösningsmetod tagits fram för jämförelse av lösningar och beräkningstider med den linjära heltalsmodellen.

4.1 Varför linjär heltalsoptimering?

Att optimera ridskolans schema kan vara svårt av olika anledningar. Ett grundläggande problem är att uppgiften blir svåröverskådlig eftersom drygt 600 elever och 40 hästar ska behandlas individuellt, och hänsyn måste tas till många samverkande faktorer. Det kan vara svårt att bedöma hur stora förbättringar som är möjliga jämfört med ett givet schema, det vill säga hur väl en viss lösning mäter sig med den som är optimal i något avseende.

Med detta som bakgrund kan konstateras att linjär optimering har en klar fördel jämfört med många andra metoder. Lösningsalgoritmen för linjära heltalsproblem bygger på att tillfälligt släppa heltalskriterierna och istället lösa ett kontinuerligt problem (se avsnitt 2.4). Därmed kan en undre gräns för målfunktionens värde bestämmas, med vilken existerande heltalslösningar kan jämföras. Skillnaden – *heltalsgapet* – mellan den bästa heltalslösningen och denna undre gräns kan användas som ett mått på hur långt ifrån optimalitet denna heltalslösning är. På så vis ger lösningsmetoden oss i teorin en bättre översikt över problemet.

En annan fördel med linjär heltalsprogrammering är att branch-and-bound-metoden garanterar att globalt optimum hittas (se teoriavsnittet 2.4 för en mer utförlig beskrivning).

Tanken med den lösningsmetod som använts är alltså att specificera alla indata och sedan optimera ridskolans schema i sin helhet för att nå global optimalitet.

4.2 Lösningsgång och modellens utformning

I allmänhet har modellen utformats för att kunna återskapa schemakriterierna på Billdals ridskola, men i flera fall är den mer generellt formulerad, vilket möjliggör lösning av andra, liknande problem. Dock måste avgränsningar göras för vilka beslut som ska fattas av människor och vad som ska ingå i optimeringen. De avgränsningar som har gjorts presenteras kortfattat nedan.

Varje lektion ska tilldelas en tid och plats, under begränsningen att alla ska få plats i ridhuset. I dagsläget finns plats för två parallella lektioner, men detta kan förstås variera och behöver därför formuleras mer allmänt. Vi har antagit att indatan till problemet innehåller en lista över *tidsblock* som är konstruerad så att lokalerna räcker till, det vill säga så att högst två tidsblock pågår samtidigt i detta fall. Ett tidsblock definieras av start- och sluttid samt veckodag.

I praktiken är det ganska enkelt att dela in veckoschemat i tidsblock, eftersom dagens första lektion bör starta i sådan tid så att eleverna, typiskt skolungdomar, har slutat skolan för dagen. Därefter fylls schemat på med tidsblock separerade av korta pauser för häst- och gruppbyte. De flesta lektioner är 45 minuter långa, och därför behöver det inte bestämmas på förhand vilka lektionstyper de olika tidsblocken ska ha. Undantag är exempelvis hoppningslektionerna som tar 60 minuter och behöver längre tidsblock. Förslagsvis placeras ett fåtal 60-minutersblock ut i schemat för att möjliggöra hoppningslektioner. Lektioner får placeras ut på tidsblock som är *minst* så långa som lektionen, vilket medför att andra lektioner kan placeras ut på tomma 60-minutersblock. Skillnaden i ett sådant fall blir att en ovanligt lång paus uppstår mellan två lektioner i schemat.

En annan möjlighet som utvärderats är att låta tidsblockens placering vara fria variabler för optimering, vilket eventuellt kan leda till att kompaktare, effektivare scheman produceras. Att släppa lektionstiderna fria visade sig dock öka lösningstiden alltför mycket i jämförelse med de förbättringar som eventuellt kan åstadkommas.

Slutligen krävs även data för när elever och lärare är tillgängliga, vilka hästar som finns och vilken maxbelastning hästarna har. Vid lösning av schemaläggningsproblemet antas att

Tabell 2: Förklaring av namngivning för variabler, parametrar och mängder.

Svenska	Engelska	Beteckning
elev	pupil	p
önskemål	request	r
häst	horse	h
tidsblock	time slot	s
(lektions)typ	type	t
lärare	teacher	θ
tillåten	allowed	a
dag	day	d
sekvens	sequence	σ
kollision	collision	c
övre gräns	upper limit	Λ
undre gräns	lower limit	λ
vikt	weight	w

ridskolans personal tilldelat en lektionstyp till varje elev som söker en plats. Inom ramarna som presenterats i detta avsnitt kan alla tänkbara fördelningar av elever, hästar och lärare representeras av modellen.

4.3 En matematisk formulering av kriterierna

I detta avsnitt presenteras de parametrar (indata) och variabler samt ekvationer och olikheter som tillsammans representerar kriterierna för schemat. Samtliga kriterier ska betraktas tillsammans, men för att göra modellen mer överskådlig har vi delat in listan under tre rubriker: elever och tidsblock, hästar samt lärare. Samtliga variabler tillåts anta värdena 1 eller 0, vilket representerar ja-eller-nej-beslut.

De namn som används på mängder, variabler och parametrar kommer från en engelsk motsvarighet till de begrepp som används på svenska: exempelvis betecknas tidsblocken med S efter engelskans *(time) slots*. I tabell 2 sammanfattas namngivningen.

Elever och tidsblock

Här följer en första lista över de mängder, parametrar och variabler som används för elever och tidsblock. Notera att hela modellen innehåller fler mängder, parametrar och variabler, vilka definieras på de kommande sidorna.

Mängder som används för elever, lektioner och tidsblock är

$$
\begin{array}{lcl}
S & = & \text{mängden av tidsblock, indexeras med } k \in S, \\
R & = & \text{mängden av önskemål } j, \text{ indexeras med } j \in R, \\
P & = & \text{mängden av elever, indexeras med } i \in P, \\
C & = & \text{mängden av kollisionsmängder, indexeras med } \mu \in C, \\
T & = & \text{mängden av lektionstyper } l, \text{ indexeras med } l \in T.
\end{array}
$$

Parametrar som används för elever, tidsblock och lektionstyper är

$$
\begin{array}{lcl}
a_r(j,k) & = & \begin{cases} 1, \text{ om önskemål } j \in R \text{ tillåts bokas på tidsblock } k \in S, \\ 0, \text{ annars,} \end{cases} \\
t_r(j,l) & = & \begin{cases} 1, \text{ om önskemål } j \in R \text{ avser bokning på lektion av typ } l \in T, \\ 0, \text{ annars,} \end{cases} \\
a_t(k,l) & = & \begin{cases} 1, \text{ om lektion av typ } l \in T \text{ tillåts bokas på tidsblock } k \in S, \\ 0, \text{ annars,} \end{cases}
\end{array}
$$

$$
\begin{aligned}
p(i,j) &= \begin{cases} 1, \text{ om önskemål } j \in R \text{ kommer från elev}^* \ i \in P, \\ 0, \text{ annars,} \end{cases} \\
c(\mu,k) &= \begin{cases} 1, \text{ om lektion } k \in S \text{ ingår i kollisionsmängd}^* \ \mu \in C, \\ 0, \text{ annars,} \end{cases} \\
\Lambda_t(l) &= \text{maximalt antal elever på lektion av typ } l \in T, \\
\lambda_t(l) &= \text{minimalt antal elever på lektion av typ } l \in T, \\
w_r(j) &= \text{viktparameter}^* \text{ för önskemål } j \in R.
\end{aligned}
$$

Variabler som används för elever och tidsblock är

$$
\begin{aligned}
r(j,k) &= \begin{cases} 1, \text{ om önskemål } j \in R \text{ bokas på tidsblock } k \in S, \\ 0, \text{ annars,} \end{cases} \\
t_s(k,l) &= \begin{cases} 1, \text{ om på tidsblock } k \in S \text{ bokas en lektion av typ } l \in T, \\ 0, \text{ annars.} \end{cases}
\end{aligned}
$$

Varje elev kan tänkas vilja delta vid flera lektioner i veckan. Därför betraktas ett antal *önskemål* om att delta vid något tidsblock av en viss lektionstyp som eventuellt kan bokas in på ett tidsblock. Varje önskemål tillhör en elev, och varje elev kan ha flera önskemål. Variabeln $r(j,k)$ indikerar om önskemål j bokas på tidsblock k.

Variabeln $t_s(k,l)$ indikerar om en lektion av typ l bokas på tidsblock k. Varje tidsblock ska tilldelas maximalt en lektionstyp, vilket modelleras med olikheterna

$$\sum_{l \in T} t_s(k,l) \leq 1, \qquad k \in S. \tag{4}$$

Att ingen lektionstyp tilldelas betyder att tidsblocket inte används. Begränsningar finns på vilka lektionstyper som ett givet tidsblock får tilldelas, vilket modelleras med ekvationen

$$\sum_{k \in S} \sum_{l \in T} t_s(k,l)\,(1 - a_t(k,l)) = 0. \tag{5}$$

Varje önskemål kan bokas in maximalt en gång, och endast på vissa tillåtna tidsblock, det vill säga då eleven är tillgänglig. Detta formuleras, analogt med (4) och (5), med olikheterna (6) och ekvationen (7).

$$\sum_{k \in S} r(j,k) \leq 1, \qquad j \in R \tag{6}$$

$$\sum_{j \in R} \sum_{k \in S} r(j,k)\,(1 - a_r(j,k)) = 0 \tag{7}$$

Tidsblock ska ha samma lektionstyp som inbokade önskemål, det vill säga om önskemål j bokas på tidsblock k måste önskemålet och tidsblocket ha samma lektionstyp. Notera att varje önskemål har *en* typ, så att $\sum_{l \in T} t_r(j,l) = 1 \ \forall j \in R$. Detta formuleras med olikheterna:

$$\sum_{l \in T} t_r(j,l) t_s(k,l) \geq r(j,k), \qquad j \in R, \ k \in S. \tag{8}$$

För varje lektionstyp finns ett maximalt och ett minimalt tillåtet antal elever, vilket formuleras med olikheterna:

$$\sum_{j \in R} r(j,k) \leq \sum_{l \in T} t_s(k,l) \Lambda_t(l), \qquad k \in S, \tag{9}$$

$$\sum_{j \in R} r(j,k) \geq \sum_{l \in T} t_s(k,l) \lambda_t(l), \qquad k \in S. \tag{10}$$

*Dessa parametrar förklaras i närmare detalj där de används nedan.

Om en elev har fler än ett önskemål ska dubbelbokning förhindras. Parametern $p(i,j)$ är en lista över alla elever i. Elementet $p(i,j)$ indikerar om önskemål j kommer från elev i.

Vi definierar en *kollisionsmängd* som en mängd tidsblock som kolliderar, så att deltagande på något av tidsblocken omöjliggör deltagande vid de övriga blocken. Parametern $c(\mu,k)$ är en lista över alla[1] kollisionsmängder, där $c(\mu,k)=1$ om tidsblock k ingår i kollisionsmängd μ, annars $c(\mu,k)=0$. Följande olikheter förhindrar då dubbelbokningar av elever:

$$\sum_{k\in S}\sum_{j\in R} c(\mu,k)r(j,k)p(i,j) \le 1, \qquad i\in P,\ \mu\in C. \tag{11}$$

Vi inför också en ekvation som egentligen är överflödig men i praktiken har visat sig förkorta lösningstiden kraftigt i vissa fall. Den säger att det maximala antalet lektioner av en typ l inte ska vara större än att alla önskemål som rör denna lektionstyp räcker till att fylla lektionerna åtminstone till den minimala gränsen. Med andra ord får inga tomma eller underbefolkade lektioner läggas in i schemat:

$$\sum_{k\in S} t_s(k,l)\lambda_t(l) \le \sum_{j\in R} t_r(j,l), \qquad l\in T. \tag{12}$$

Hästar

Här följer ytterligare några mängder, parametrar och variabler som krävs för att beskriva kriterierna för hästar.

Mängder som används för hästar är

H	=	mängden av hästar, indexeras med $m\in H$,
Σ	=	mängden av sekvenser, indexeras med $\nu\in\Sigma$,
D	=	mängden av dagskombinationer, indexeras med $\xi\in D$.

Parametrar som används för hästar är

$t_h(m,l)$	=	$\begin{cases} 1, \text{ om häst } m\in H \text{ tillåts bokas på lektionstyp } l\in T, \\ 0, \text{ annars,} \end{cases}$
$\Lambda_{h\sigma}(m)$	=	maximal tillåten belastning* i sekvens för häst $m\in H$,
$\Lambda_{hd}(m)$	=	maximal tillåten belastning* per dag för häst $m\in H$,
$w_{t\sigma}(l)$	=	belastning i sekvens av lektionstyp $l\in T$,
$w_{td}(l)$	=	belastning i dagskombination av lektionstyp $l\in T$,
$\sigma(\nu,k)$	=	$\begin{cases} 1, \text{ om tidsblock } k\in S \text{ ingår i sekvens* } \nu\in\Sigma, \\ 0, \text{ annars,} \end{cases}$
$d(\xi,k)$	=	$\begin{cases} 1, \text{ om tidsblock } k\in S \text{ ingår i dagskombination* } \xi\in D, \\ 0, \text{ annars,} \end{cases}$
Ω	=	stort tal.

Variabeln som används för hästar är

$$h(m,k) = \begin{cases} 1, \text{ om häst } m\in H \text{ bokas på tidsblock } k\in S, \\ 0, \text{ annars.} \end{cases}$$

På varje tidsblock ska lika många hästar som elever bokas in, vilket formuleras med ekvationerna

$$\sum_{m\in H} h(m,k) = \sum_{j\in R} r(j,k), \qquad k\in S. \tag{13}$$

[1] I praktiken behöver vi bara betrakta de största möjliga kollisionsmängderna, det vill säga om en kollisionsmängd är en äkta delmängd av någon annan, behöver den mindre mängden inte tas med i parametern c.

*Dessa parametrar förklaras i närmare detalj där de används nedan.

För hästarna finns begränsningar för vilka lektionstyper som är tillåtna eftersom inte alla hästar kan delta vid lektioner av varje lektionstyp. Detta kriterium formuleras med olikheterna

$$\sum_{l \in T} t_s(k,l) t_h(m,l) \geq h(m,k), \qquad m \in H,\ k \in S. \tag{14}$$

Hästar får inte dubbelbokas. Vi skriver analogt med ekvation (11) för eleverna:

$$\sum_{k \in S} h(m,k) c(\mu,k) \leq 1, \qquad m \in H,\ \mu \in C. \tag{15}$$

Hästarnas totala belastning begränsas på två sätt: belastning av flera lektioner i direkt följd och total belastning per dag. För detta ändamål finns parametrarna $w_{t\sigma}$ och w_{td} som anger belastningen av olika lektionstyper på hästarna, då vi räknar i *sekvens* respektive *dagskombination*.

Vi definierar en *sekvens* som en mängd tidsblock som ligger i följd, så att deltagande på samtliga är tidsmässigt möjligt. Parametern σ är en lista över alla[2] sekvenser ν sådana att $\sigma(\nu,k) = 1$ om tidsblock k ingår i sekvens ν, annars är $\sigma(\nu,k) = 0$.

På motsvarande sätt definierar vi en *dagskombination* som en mängd tidsblock som ligger på samma dag men inte kolliderar, så att deltagande på samtliga är tidsmässigt möjligt. Parametern d är en lista över alla[3] dagskombinationer ξ så att $d(\xi,k) = 1$ om tidsblock k ingår i kombinationen ξ, annars är $d(\xi,k) = 0$.

Olikheterna (16) begränsar den totala belastningen i sekvens endast om hästen bokas in på samtliga tidsblock i sekvensen:

$$\sum_{k \in S} \sum_{l \in T} \sigma(\nu,k) t_s(k,l) w_{t\sigma}(l) \leq \Lambda_{h\sigma}(m) + \Omega \sum_{k \in S} \sigma(\nu,k) \left(1 - h(m,k)\right), \qquad \nu \in \Sigma,\ m \in H. \tag{16}$$

Olikheterna (17) är analogt formulerade för dagskombinationer:

$$\sum_{k \in S} \sum_{l \in T} d(\xi,k) t_s(k,l) w_{td}(l) \leq \Lambda_{hd}(m) + \Omega \sum_{k \in S} d(\xi,k) \left(1 - h(m,k)\right), \qquad \xi \in D,\ m \in H. \tag{17}$$

Lärare

Slutligen inför vi en mängd, en variabel och några parametrar som används för att beskriva kriterier för lärare.

Mängden som används är

Θ = mängden av lärare, indexeras $n \in \Theta$.

[2]Vi är i praktiken bara intresserade av de sekvenser som kan begränsa schemaläggningen. Därmed behöver endast sekvenser som, beroende på lektionstyp, *kan* ge för hög belastning för någon häst tas med i parametern σ – kortare sekvenser kan utelämnas. De längsta sekvenser som behöver tas med är de *kortaste* som garanterat ger för hög total belastning, oavsett lektionstyper och häst – ty varje sekvens som är längre kommer att ha en sådan begränsande sekvens som äkta delmängd och behöver därför inte betraktas explicit. Om exempelvis alla hästar klarar av två lektioner i rad, och ingen häst klarar mer än fyra lektioner i rad, räcker det att σ innehåller alla sekvenser med 3–5 tidsblock.

[3]Se ovanstående fotnot om sekvenser. Listan över dagskombinationer kan i praktiken reduceras på samma sätt.

Parametrar som används för lärare är

$$\begin{aligned} a_\theta(n,k) &= \begin{cases} 1, \text{ om lärare } n \in \Theta \text{ tillåts bokas på tidsblock } k \in S \text{ (läraren är tillgänglig),} \\ 0, \text{ annars,} \end{cases} \\ \Lambda_{\theta\sigma}(n) &= \text{maximalt antal lektioner i sekvens för lärare } n \in \Theta, \\ \Lambda_{\theta w}(n) &= \text{maximalt antal lektioner per vecka för lärare } n \in \Theta, \\ \Omega &= \text{stort tal.} \end{aligned}$$

Variabeln som används för lärare

$$\theta(n,k) = \begin{cases} 1, \text{ om lärare } n \in \Theta \text{ bokas på tidsblock } k \in S, \\ 0, \text{ annars.} \end{cases}$$

En lärare ska bokas in på varje tidsblock som har tilldelats en lektionstyp, alltså om en lektion ska hållas vid denna tid, enligt

$$\sum_{n\in\Theta} \theta(n,k) = \sum_{l\in T} t_s(k,l), \qquad k \in S. \tag{18}$$

Lärarna får endast bokas in på tillåtna tider, det vill säga då de är tillgängliga. De får heller inte dubbelbokas. Analogt med ekvationerna (7) och (11) för elever formuleras detta med ekvationen (19) och olikheterna (20):

$$\sum_{n\in\Theta}\sum_{k\in S} \theta(n,k)\,(1 - a_\theta(n,k)) = 0, \tag{19}$$

$$\sum_{k\in S} \theta(n,k)c(\mu,k) \le 1, \qquad n \in \Theta,\ \mu \in C. \tag{20}$$

Det totala antalet lektioner per vecka för lärare begränsas av parametern $\Lambda_{\theta w}(n)$. Vi skriver således

$$\sum_{k\in S} \theta(n,k) \le \Lambda_{\theta w}(n), \qquad n \in \Theta. \tag{21}$$

Varje lärare har också ett maximalt tillåtet antal lektioner i direkt följd, $\Lambda_{\theta\sigma}(n)$. Detta krav formuleras analogt med olikheterna (16), som

$$\sum_{k\in S}\sum_{l\in T} \sigma(\nu,k)t_s(k,l) \le$$

$$\Lambda_{\theta\sigma}(n) + \Omega \sum_{k\in S} \sigma(\nu,k)\,(1 - \theta(n,k))\,, \qquad n \in \Theta,\ \nu \in \Sigma. \tag{22}$$

4.4 Andra varianter av modellen

Ovanstående kriterier kan användas för att lösa schemaläggningsproblemet på ridskolan, där bland annat önskemål från elever och lärare, maxbelastning på hästar och tillgängliga lokaler kan kombineras till ett komplett schema. Flera variationer av problemet kan dock tänkas.

Ett viktigt exempel är det fall då ett lektionsschema för elever och lärare är klart, men tillgången på hästar varierar på grund av skada eller sjukdom. Detta reducerade problem har ägnats särskild uppmärksamhet, och resultat från flera olika testkörningar återfinns i avsnitt 5.

En metod för att undersöka ett sådant reducerat problem där vissa variabler bestämts på förhand är att göra om variablerna i fråga till parametrar och sedan ta bort eventuella överflödiga ekvationer. Det kan också bli aktuellt att formulera en ny målfunktion vid körning av ett reducerat problem.

4.5 Målfunktioner

Flera möjliga kvalitetsmått på schemat har identifierats, exempelvis

Utnyttjande av resurser. Ridskolan finns till för att hålla lektioner, så det är ett grundläggande önskemål att de möjliga lektionstillfällena ska fyllas med många elever.

Bekvämlighet och effektivitet. För lärarna är det önskvärt att slippa onödiga väntetider. Schemat bör vara kompakt, så att lektionerna inte pågår sent på kvällen i onödan.

Rättvisa. Det finns ingen särskild anledning att exempelvis vissa hästar ska arbeta oproportionerligt lite på bekostnad av andra. Om två lärare har lika önskemål om arbetstider finns ingen anledning att den ena ska få ett bekvämare schema än den andra.

Den målfunktion som har använts i de flesta försöken är maximering av antalet inbokade önskemål, viktat med parametern w_r:

$$\text{maximera} \sum_{j \in R} \sum_{k \in S} r(j,k) w_r(j).$$

Vi har låtit $w_r(j) = 1 \, \forall j \in R$, men den kan sättas till något annat värde för vissa elever. Om exempelvis förra terminens elever som vill fortsätta har högre prioritet än nybörjare, kan viktparametern sättas till ett lägre värde för de önskemål som rör nybörjarlektioner.

Ett annat villkor som har använts är jämvikt mellan hästarnas belastning. Som ett mått på den totala belastningen per häst och vecka används beteckningen

$$b(m) = \sum_{k \in S} h(m,k),$$

det vill säga antalet lektioner per vecka för häst m. I ett rättvist schema bör $b(m)$ inte variera för mycket mellan hästarna. Vi vill därför minimera variansen

$$\begin{aligned} \text{Var}\,(b) &= \sum_{m \in H} \left(b(m) - \langle b \rangle\right)^2 = \left\langle b^2 \right\rangle - \langle b \rangle^2 \\ &= \frac{1}{N_h} \sum_{m \in H} \left(\sum_{k \in S} h(m,k) \right)^2 - \left(\frac{1}{N_h} \sum_{m \in H} \sum_{k \in S} h(m,k) \right)^2, \end{aligned}$$

där N_h är det totala antalet hästar.

Notera att denna funktion inte är linjär, utan kvadratisk. Det kvadratiska kriteriet kan dock hanteras av CPLEX i detta fall. Se avsnitt 2.2 för en kort teoretisk bakgrund.

Märk också väl att en triviallösning i form av ett tomt schema är den optimala lösningen med avseende på detta kriterium – då är belastningen $b(m)$ och dess varians lika med noll. Därför linjärkombinerar vi de två ovanstående kriterierna enligt

$$\text{maximera} \sum_{k \in S} \sum_{j \in R} r(j,k) w_r(j) - \omega \text{Var}\,(b)\,. \tag{23}$$

Därmed görs en avvägning mellan de två kriterierna (maximalt antal elever respektive minimal varians), som bestäms av parametern ω. Om vi låter ω vara litet kommer minimering av $\text{Var}\,(b)$ att bli ett sekundärt mål, så att rättvisa mellan hästarna aldrig kommer att sättas före maximering av antalet elever. Parametern ω är alltså ett mått på det relativa värdet av några fler elever jämfört med jämnare arbetsfördelning mellan hästarna. Detta resonemang om avvägningar är allmänt giltigt och kan alltid tillämpas då olika målfunktioner linjärkombineras.

4.6 Heuristisk algoritm

Förutom att lösa det linjära heltalsproblemet med hjälp av befintliga linjära lösningsmetoder (CPLEX) har också en heuristisk lösningsmetod tagits fram. I teoriavsnittet (avsnitt 2.5) ges en bakgrund till nyttan av en sådan metod. I detta avsnitt beskrivs idén bakom den heuristiska metodens tanke och dess implementering.

Det finns många sätt att utforma en heuristisk metod och utvecklingen av en sådan hade varit tillräckligt underlag för ett separat arbete. Syftet med den framtagna heuristiska

algoritmen är att fungera som en grund för jämförelse för lösningar av den linjära modellen. Således ligger fokus på att producera godtagbara lösningar på kort tid. Tanken med den heuristiska algoritmen är att i så hög grad som möjligt utnyttja den uppenbara strukturen i problemet. Till exempel är det onödigt att genomsöka alla elever då ett tidsblock av en viss typ skall fyllas med elever. Överväganden av denna typ har kombinerats med slumpinslag för att söka igenom en bred uppsättning lösningskandidater. Algoritmen bygger på den linjära heltalsmodellen i den utsträckning att huruvida en elev, häst eller lärare tillåts bokas på ett tidsblock avgörs utifrån samma kriterier som denna modell. Nedan följer en kortfattad beskrivning av den heuristiska metoden.

```
bestäm en målfunktion
dela in elever och hästar efter tillåtna grupptyper
while (målfunktionen är under målvärdet)
    sortera elever, hästar, lärare och tidsblock slumpmässigt
    for (tidsblock)
        välj slumpmässig lektionstyp för detta tidsblock
    end
    for (tidsblock)
        for (alla elever som matchar tidsblockets lektionstyp)
            om möjligt utan att bryta något villkor, boka eleven på tidsblocket
            if (tidsblocket är fyllt med elever)
                break
            end
        end
        for (alla hästar som kan delta vid tidsblockets lektionstyp)
            om möjligt utan att bryta något villkor, boka hästen på tidsblocket
            if (en häst är inbokad för varje elev)
                break
            end
        end
        for (lärare)
            om möjligt utan att bryta något villkor, boka läraren på tidsblocket
            if (en lärare är inbokad)
                break
            end
        end
        if (tidsblocket inte fyllt till undre gränsen för
                antal elever, hästar och lärare)
            töm tidsblocket på elever, hästar och lärare
        end
    end
    for (alla tomma tidsblock)
        välj en ny lektionstyp som inte testats tidigare för tidsblocket
        upprepa ovanstående procedur för elever, hästar och lärare
    end
    if (det producerade schemat har bäst målfunktionsvärde hittills)
        spara schemat
    end
end
```

5 Resultat

De modeller och lösningsmetoder som presenteras ovan har använts för att simulera en rad scenarier. Dessa scenarier har tagits fram för att svara på frågor som bedöms ha betydelse för schemaläggningsproblemet. I detta avsnitt presenteras resultaten av dessa testkörningar.

Den linjära heltalsmodellen som beskrivs i avsnitt 4.3 har implementerats i AMPL och CPLEX. Samtliga CPLEX-beräkningar har körts på en 8xDual Core AMD Opteron(tm) Processor 875 vid 1000MHz med 8GB RAM. Den heuristiska algoritmen har implementerats i MATLAB och körts på en Intel Core Duo 2GHz med 2GB RAM.

5.1 Befintligt schema

Ett intressant test är att ta reda på huruvida det av ridskolan framtagna schemat är giltigt enligt de i modellen ställda kraven. För att undersöka detta kördes modellen med fixerade lektionstyper enligt det befintliga schemat som inparameter. Som elevunderlag i detta fall användes samma mängd elever (605 stycken) som ridskolan i dagsläget undervisar varje vecka. Eleverna antogs vara tillgängliga samtliga lektionstillfällen och alla hästarna antogs vara i god form. Lärarna antogs också vara tillgängliga vid samtliga tillfällen. Målet var att placera samtliga elever i schemat. Detta gav resultatet i tabell 3.

Tabell 3: Resultat av testkörning med det befintliga schemat och maximalt 605 elever. Som jämförelse för lösningen i CPLEX presenteras en lösning framtagen av den heuristiska algoritmen. I tabellen anges avvikelsen från det verkliga antalet inbokade elever.

	Antal elever	Avvikelse	Beräkningstid [s]	Visad optimalitet
Linjär heltalsmodell	601	0.7%	506	Ja
Heuristik	593	2.0%	1000	Nej

Resultatet visar att en optimal lösning av problemet innebär schemaläggning av 601 elever. Att hålla lektioner för 605 elever är alltså inte möjligt under de givna förutsättningarna. Resultatet visar att den heuristiska lösningsmetoden inte producerade en optimal lösning inom en rimlig tid.

5.2 Skadade hästar

Då ett schema tas i bruk är det viktigt att det inte fullständigt kollapsar så fort en häst blir skadad. Enligt personal på Billdals ridklubb är i genomsnitt 20% av alla hästar samtidigt helt eller delvis ur drift. Det vill säga att ett visst antal hästar är helt skadade och inte kan användas alls och ett visst antal är delvis skadade och endast kan belastas lätt. Dessutom förekommer utbildning av hästar vilket även detta gör dem otillgängliga för lektioner.

Skadors inverkan på ett befintligt schema kan undersökas genom att upprepade gånger slumpmässigt simulera att 10% av hästarna är helt skadade och 10% av hästarna är delvis skadade, så att de endast får belastas till 50% av den ursprungliga maxbelastningen. Detta test har utförts tio gånger med slumpvis valda helt eller delvis skadade hästar, och i övrigt samma förutsättningar som testet i avsnitt 5.1. Resultatet presenteras i tabell 4.

Tabell 4: Resultat av upprepade testkörningar med den linjära heltalsmodellen där 20% av hästarna är helt eller delvis skadade. Utgångspunkten är det befintliga schemat och 605 elever. I tabellen anges avvikelsen från det antalet elever som är inbokade i verkligheten.

	Antal elever	Avvikelse	Visad optimalitet
Linjär heltalsmodell	590	2.3%	Ja
"	569	6.0%	Ja
"	592	2.1%	Ja
"	592	2.1%	Ja
"	588	2.8%	Ja
"	591	2.3%	Ja
"	590	2.3%	Ja
"	588	2.8%	Ja
"	591	2.3%	Ja
"	592	2.1%	Ja

Då alla hästar är friska är det optimala antalet elevplatser 601. Genomsnittet av resultatet i tabell 4 är 588, vilket är 2% färre än då alla hästar antas vara friska.

5.3 Ökad belastning på hästar

En ständig fråga då schemat skall läggas är hur hårt hästarna kan belastas. Det lätt att inse att hårdare belastning av hästarna gör det möjligt att tillgodose fler elevers önskemål. Att öka trycket på hästarna kan dock ha negativa effekter om detta till exempel ökar antalet skador.

I modellen är gränser satta för att se till att hästarna inte överbelastas. För att få en förståelse för hur hårda dessa restriktioner är i förhållande till ett verkligt scenario är det illustrativt att undersöka vad som händer om gränserna tänjs. Detta scenario är tänkt att maximera antalet elevplatser utan att bryta mot restriktionerna för hästbelastning. Detta gjordes genom att skapa en schemamall utifrån det befintliga schemat där ett överflöd av tidsblock lagts till. Sedan maximerades antalet elevplatser utan att ta hänsyn till det verkliga elevunderlaget eller lärare. Målvärdet för denna optimering är okänt.

Tabell 5: Resultat av det scenario där restriktioner på hästbelastning undersöks. Hästar placeras ut bland ett överflöd av lektioner utan att ta hänsyn till elever eller lärare.

	Antal elever	Visad optimalitet
Linjär heltalsmodell	Slut på minne	-
Heuristik	678	Nej

I tabell visas resultatet av detta test. Den heuristiska metoden producerade ett schema med utrymme för 678 elever. Detta är 13% mer än vad schemat i dagsläget har utrymme för då samtliga hästar är friska. Resultatet kan ses som en under gräns för vad som är möjligt, eftersom ett eventuellt optimalt värde garanterat är högre än resultatet i detta test. Då den linjära heltalsmodellen användes vid detta försök resulterade detta i att minnesutrymmet tog slut.

5.4 Utvidgning av befintligt schema

I dagsläget finns det en kö av nybörjare till ridskolan som önskar börja rida. Att tillåta ett antal extra nybörjargrupper är vad detta scenario simulerar. Som tidigare konstaterats gäller att det befintliga schemat endast har utrymme för 601 elever. Elevunderlaget utökades med

fyra extra nybörjargrupper. Dessutom utvidgades det befintliga schemat med motsvarande tomma tidsblock. Resultatet av testkörningen presenteras i tabell 6.

Tabell 6: Resultat av scenario där det befintliga schemat med 605 elever utvidgats för att tillåta 4 extra nybörjargrupper med 12 elever i varje, det vill säga totalt 653 elever.

	Antal elever	Visad optimalitet
Linjär heltalsmodell	637	Ja

Detta försök visar att det finns utrymme för ridskolan att undervisa fler nybörjare. Jämfört med det schemat som den linjära heltalsmodellen beräknat (601 elever) utvidgas antalet platser med 6%. Observera att det genomsnittliga lektionsantalet per häst ökar lika mycket i relativa mått.

5.5 Lösning av generellt problem

Med detta försök undersöktes möjligheten att lösa problemet på generell form, det vill säga då indata är sådan att stor frihet lämnas åt optimeringen. En schemamall med tidsblock placerade på samma vis som i det befintliga schemat användes. Parametern a_t, som anger vilka lektionstyper som är möjliga på de olika tidsblocken, valdes så att 4–6 rimliga lektionstyper tilläts på varje tidsblock. Målet var att maximera antalet inbokade elever. Resultatet av detta test visas i tabell 7.

Tabell 7: Resultat av lösning av ett generellt schemaläggningsproblem. Totalt kunde 605 elever bokas in. I tabellen anges avvikelsen från detta maximala antal.

	Beräkningstid [s]	Antal elever	Avvikelse	Visad optimalitet
Linjär heltalsmodell	5400	590	2.5%	Nej
"	18000	596	1.5%	Nej
"	54000	596	1.5%	Nej
Heuristik	1	586	3.1%	Nej
"	30	602	0.5%	Nej
"	5400	602	0.5%	Nej

I tabell 7 observeras en tydlig skillnad i lösningstider mellan optimering i CPLEX och användning av den heuristiska algoritmen.

5.6 Varians av hästbelastning

I testscenarierna fram till till denna punkt har målet varit att ge så många elever som möjligt lektionsplatser. I avsnitt 4.5 diskuteras alternativa målfunktioner och nyttan av dessa. Det bedömdes vara intressant att fördela belastningen lika mellan de hästar som är tillgängliga. En målfunktion som strävar mot precis detta presenteras i (23).

För att skapa en uppfattning av hur mycket variationen i hästarnas veckobelastning kan minska, jämfört med då ingen hänsyn tas till denna, följer en jämförelse med testet i avsnitt 5.1. För schemat som presenteras i tabell 3 är variansen för hästarnas veckobelastning $\mathrm{Var}(b) = 19.9$. I tabell 8 visas resultatet av optimering där målfunktionen i ekvation (23) använts för några olika värden på viktparametern ω.

Tabell 8: Resultat av optimering där målet är en viktad summa av att maximera antalet elever och minimera standardavvikelsen mellan hästarnas veckobelastning. Förutom målfunktionen är förutsättningarna desamma som för testet i avsnitt 5.1.

	ω	Var(b)	Antal elevplatser	Beräkningstid [s]	Visad optimalitet
Linjär heltalsmodell	0.1	2.12	601	10860	Nej
"	1	1.90	590	10860	Nej

I tabell 8 syns att en avsevärd minskning av variansen i hästbelastningen är möjlig om detta är ett optimeringsmål. Resultatet påvisar också avvägningen mellan olika optimeringsmål. Då $\omega = 0.1$ får det optimala antalet elever lektionsplatser, vilket visades vara 601 i avsnitt 5.1. Då $\omega = 1$ prioriteras bokning av ett antal elever bort för att minska variansen ytterligare.

6 Diskussion och analys

I detta avsnitt diskuteras dagens fungerande schema jämfört med ett beräknat optimalt schema enligt de uppställda kraven och villkoren. Kärnfrågan är hur relevant en matematiskt optimal lösning är för detta specifika schemaläggningsproblem. Scenarier med de alternativa förutsättningarna som finns beskrivna och testade i resultatdelen behandlas också. Vidare diskuteras alternativa lösningsmetoder för både detta enskilda problem samt liknande problem inom liknande och andra områden. I avsnittet utvärderas även styrkor, svagheter och relevans hos de valda arbetsmetoderna och den utvecklade modellen.

6.1 Ridskolans schema

Försöket som redovisas i avsnitt 5.1 visar att dagens schema i verkligheten bryter mot något av de villkor som specificeras i modellen. Förklaringen till detta är att de strikta kraven genom en mänsklig övervägning i vissa fall helt eller delvis kan försummas. Sådana överväganden bedöms vara svåra eller omöjliga att implementera fullt ut i en modell. En rekommendation till vidare studie är att närmare undersöka vilka villkor som utgör de begränsande faktorerna.

De tester som gjordes med helt eller delvis bortfall av slumpmässigt valda hästar visar att dagens schema har en viss robusthet mot förändringar i hästunderlaget. För att tillgodose samtliga elevers lektionstider betyder det att slitaget per häst ökar för de hästar som är friska. Det krävs att de friska hästarna deltar vid lektionstyper utanför sina kompetensområden eller att vissa hästar belastas över maxgränsen för att alla elever ska få rida på sin lektion. Antalet elevplatser som kunde erbjudas enligt försöket varierade då olika hästar slumpmässigt skadades. Detta kan förklaras av att vissa hästar är betydligt mer mångsidiga och därför mer betydelsefulla för schemats funktionalitet.

Resultaten i avsnitt 5.3 visar att det finns gott om möjligheter att belasta hästarna hårdare inom ramen för modellen. Att detta är möjligt samtidigt som hästunderlaget begränsar antalet elevplatser i andra test verkar vid en första blick märkligt. Förklaringen till detta är hur tidsblock, elevunderlag och lärare tillåts kombineras. Det hade alltså varit möjligt för ridskolan att utöka antalet elevplatser om elevunderlaget kunde väljas fritt, men så är inte fallet.

Ridskolan har i dagsläget kö till nybörjarplatserna och i avsnitt 5.4 antyds att fler elever kunde få plats på ridskolan, detta resultat bygger dock på att alla hästar är tillgängliga. Flera eller större brott mot de uppsatta kriterierna än i dagsläget skulle bli nödvändiga eftersom hästunderlaget varierar på grund av skada och sjukdom. Att i dag ta in fler nybörjare kan också ställa till problem i framtiden. Nybörjare blir efter en viss tid mer erfarna och skall då närvara vid lektioner av mer avancerad typ. Frågan om att ta in fler nybörjare blir alltså i förlängningen en fråga om att utvidga schemat med elever av alla typer.

Maximering av antalet elever på ridskolan är en enkel målfunktion, men den innebär att många andra kvaliteter hos schemat försummas. En intressant egenskap som har utvärderats är hur belastningen mellan hästarna fördelas (se avsnitt 5.6). Minimering av variansen av antal lektioner per häst och vecka inkluderades som sekundärt mål i optimeringen i detta försök. Resultatet blev en markant förbättring av belastningsfördelningen för hästarna.

Den modell som använts är så allmänt formulerad att den bör vara tillämpbar på andra, liknande ridskolor. Vissa av de friheter som ges i modellen har inte utforskats i projektet, exempelvis möjligheten att begränsa vilka tider som lärare och elever är tillgängliga. Om det finns önskemål som inte täcks av modellen, kan ytterligare begränsningar eller sekundära optimeringsmål införas. Till exempel är modellen inte konstruerad på ett sådant sätt att elever kan ange prioritet på lektionstider eller vilka hästar de vill ha. Lärarna kan inte heller välja vilka tider eller lektionstyper de vill ha. En stor mängd sådana önskemål är möjliga att införa, antingen som strikta krav eller sekundära optimeringsmål, men de flesta har inte kunnat implementeras inom ramen för detta projekt.

En matematisk modell anses enligt ovanstående överväganden vara ett starkt verktyg vid utarbetning av en grundlösning till ett problem av denna natur. Mänsklig finslipning och justering är dock nödvändig i praktiska fall när mjuka krav formulerats som strikta i en förprogrammerad modell. En kombination av en matematisk modell och mänsklig utvärdering

och eventuell justering, anses därför vara en bättre arbetsgång jämfört med ett rent manuellt arbete eller en rent automatiserad lösning.

6.2 Metod och modell

En stor fördel med linjära formuleringar av optimeringsproblem är att global optimalitet kan påvisas. I vårt fall garanterar branch-and-bound-algoritmen att globalt optimum hittas. Dock blir denna egenskap mindre intressant i sådana fall där beräkningstiden har nått en oacceptabel nivå och den bästa funna lösningen inte är optimal eller kan visas vara nära optimal. Därför finns en övre gräns för hur omfattande ett optimeringsproblem kan vara innan det blir ointressant att försöka formulera det linjärt. Ett vanligt problem [13] är att CPLEX får slut på arbetsminne då heltalsproblem löses eftersom viss information om varje lösningsträdet måste sparas. De kombinatoriska möjligheterna i vissa delar av trädet är mycket stora, vilket gör det svårt att begränsa minnesanvändningen.

Exakt hur fort lösningstiden ökar med storleken på problemet (antalet elever, hästar, tidsblock etc) går inte att säga. Det har visats att den ursprungliga simplexalgoritmen – för kontinuerliga problem – har exponentiell tidskomplexitet i värsta fall, men i praktiken har lösningstiden allmänt visat sig växa polynomiellt med problemets storlek [14]. Notera att simplexalgoritmen används i varje nod i lösningsträdet. Det kan åtminstone konstateras att den totala lösningstiden ökar betydligt fortare än linjärt, det vill säga att en fördubbling av antalet variabler bör resultera i betydligt mer än en fördubbling av lösningstiden. Antalet variabler ökar i sin tur mer än linjärt med ridskolans storlek: exempelvis består $r(j,k)$ av NM st binära variabler där N är antalet önskemål (elever) och M är antalet tidsblock.

Lösningstiden varierar beroende på vilken målfunktion som används. De två testkörningar som gjorts med ridskolans befintliga schema (se avsnitt 5.1 och 5.6) är identiska förutom att målfunktionen i det senare fallet även innehåller variansen av antal lektioner för hästarna. Med den enklare målfunktionen visas optimalitet efter mindre än 10 minuter, men i det andra fallet hade optimalitet inte visats då beräkningarna avbröts efter 180 minuter. Notera dock att resultatet vid denna tidpunkt var bra: lika många elever hade bokats medan variansen av hästarnas belastning sänktes betydligt.

Om det primära målet är att hitta välfungerande schemalösningar kan det vara praktiskt att använda någon heuristisk lösningsmetod. Det enkla heuristikförslag som har utvärderats i detta projekt gav generellt resultat jämförbara med CPLEX. I avsnitt 5.1 och 5.4 finns exempel på att CPLEX presterar bättre än den heuristiska algoritmen, och i avsnitt 5.5 finns exempel på det motsatta. Jämförelse av de två metoderna är dock vansklig eftersom CPLEX implementerats i kompilerad kod och har körts på en kraftfull dator, medan den heuristiska algoritmen implementerats i MATLAB och körts på en mindre kraftfull dator. Vi drar ändå slutsatsen att vår heuristiska metod har jämförbar prestanda och dessutom kan implementeras i något programspråk som får användas gratis till skillnad från CPLEX.

En tänkbar förbättring är att dela upp problemet i flera mindre delproblem. De olika delarna i modellen är kopplade till varandra, men kan separeras i delar som inte är så starkt beroende av varandra, exempelvis 1) gruppering av elever och tilldelning av tidsblock, 2) tilldelning av hästar till elevgrupper och 3) tilldelning av lärare till elevgrupper. Det finns olika metoder, exempelvis Dantzig-Wolfe-dekomponering [15], som kan användas för att lösa stora optimeringsproblem som delas upp i flera kopplade delar. Ett exempel ur verkligheten ges av Andersson och Värbrand [16] som har tillämpat Dantzig-Wolfe-dekomponering för att snabbt lösa optimeringsproblem där det krävs schemaändring av flygtrafik. Någon sådan metod bedöms vara ett gångbart alternativ för att lösa den mest generella varianten av problemet.

7 Slutsats

Vi bedömer att det befintliga schemat är förhållandevis välgjort, det vill säga att inga stora förbättringar är möjliga utan att någon resurs belastas hårdare än kriterierna medger. Med diskussionen i avsnitt 6.1 som underlag konstateras att det finns möjliga utvidgningar av schemat, men dessa leder till ökad belastning av framförallt hästar. Huruvida detta är godtagbart är en bedömningsfråga som inte kan besvaras av en linjär heltalsmodell. Lösningarna måste alltså analyseras innan de används, på samma sätt som ridskolans personal idag bedömer olika beslut i schemaläggningen.

Den linjära heltalsmodell som presenteras i denna rapport har flera fördelar, vilka diskuteras i avsnitt 6.2. Bland annat garanterar branch-and-bound-algoritmen att den slutliga lösningen är globalt optimal. Denna egenskap kan användas för att analysera problemet och identifiera begränsande faktorer. Kunskap av denna typ kan utnyttjas för att med minsta resursutvidgning öka till exempel ridskolans inkomst eller minska hästarnas belastning.

Vår bedömning är att ridskolan kan dra nytta av datorbaserad optimering vid schemaläggning, antingen linjär heltalsprogrammering eller en vidareutvecklad heuristisk algoritm. Lösningarna kan fungera som förslag då ett nytt schema skall tas fram. Utifrån ett sådant förslag kan personalen sedan göra justeringar efter övervägning av vad som faktiskt är gångbart i verkligheten. Därmed kan fördelarna med en dators beräkningskraft kombineras med en människas intuitiva förmåga.

Referenser

[1] S.M. Sinha. *Mathematical Programming: Theory and Methods*. New Delhi: Elsevier, 2006.

[2] George B. Dantzig. Reminiscences about the origins of linear programming. *Operations Research Letters*, 1(2):43 – 48, 1982.

[3] George B. Dantzig. Linear programming. *Operations Research*, 50(1):42 – 47, 2002.

[4] P.H. Williams. *Model Building in Mathematical Programming*. Chichester: John Wiley & Sons, Ltd., 4e utgåvan, 2008. pp. 3–5, 17, 18, 59–65, 129, 130, 144, 145.

[5] R. Borndörfer, A. Löbel och S. Weider. *Computer-aided Systems in Public Transport*, kapitel A Bundle Method for Integrated Multi-Depot Vehicle and Duty Scheduling in Public Transport. Berlin: Springer, 2008.

[6] Edmund Kieran Burke och Sanja Petrovic. Recent research directions in automated timetabling. *European Journal of Operational Research*, 140(2):266 – 280, 2002.

[7] A. Schaerf. A survey of automated timetabling. *Artificial Intelligence Review*, 13(2), 1999.

[8] S. Nash och A. Sofer. *Linear and Nonlinear Programming*. New York: McGraw-Hill, 1996. pp. 70–74, 76–78, 94–101.

[9] P.H. Williams. *Logic & Integer Programming*, band 130 av *International Series in Operations Research & Management Science*. London: Springer, 2009. pp. 25–49.

[10] J. Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. New York, Addison-Wesley, 1983. pp. 3, 4, 14–16.

[11] R. Fourer, D.M. Gay och B.W. Kernigham. *AMPL: A Modeling Language for Mathematical Programming*. Duxbury Press, 2:a utgåvan, 2002.

[12] IBM leadership in optimization. [hämtat 2010–03–04], tillgänglig: `http://www-01.ibm.com/software/websphere/products/optimization/leadership/`.

[13] *User's Manual for CPLEX: IBM ILOG CPLEX V12.1*, 2009. [hämtat 2010–03–04], tillgänglig: `ftp://public.dhe.ibm.com/software/websphere/ilog/docs/optimization/cplex/ps_usrmancplex.pdf`.

[14] D.A Spielman och S-H. Teng. Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. *Computing Research Repository*, cs.DS/0111050v7, 2001.

[15] L.S. Lasdon. *Optimization Theory for Large Systems*, kapitel 3. New York: McMillan Publishing, 1970.

[16] T. Andersson och P. Värbrand. The flight perturbation problem. *Transportation Planning and Technology*, 27(2):91 – 117, 2004.