Projekt 2 i TAOP18 *Supply chain optimization*

Pedersen, Anton
antpe759@student.liu.se

Lindholm, David
davli921@student.liu.se

4 December 2016

Innehåll

1 Introduktion

Det känns mycket hedrande att vi har fått uppdraget att göra en analys över hur Statoil ska planera sina rutter för att distribuera bensin och andra produkter till sina bensinstationer. Att ha en bra ruttplanering är viktigt av tre anledningar. Den första anledningen är att transport kostar mycket pengar och genom en att minska logistikkostnader kan företagets vinst öka. Den andra anledningen är att ungefär 6 % av europas koldioxidutsläpp kommer från tungtransporter, t.ex. tankbilar och lastbilar (Road transport: Reducing CO2 emissions from vehicles", 2016). Därför är det från ett miljömässigt perspektiv väldigt viktigt att man använder sig av effektiva rutter. Slutligen kan en välgenomtänkt ruttplanering spara tid vilket gör att logistikservicen kan förbättras (kortare ledtid), vilket på sikt kan skapa nöjdare kunder.

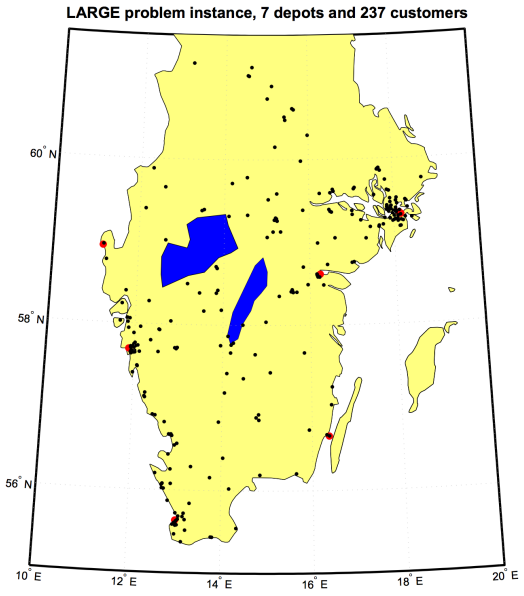
För att lösa problemet så delade vi in det i två delar:

- Del 1: Ett problem där vi tilldelar varje kund en depå. Denna tilldelning kommer att användas vid ruttplaneringen.,
- Del 2: Bestämma vilka rutter som tankbilarna ska köra.

För del 1 använde vi en enkel algoritm som beskrivs senare och för del 2 användes tabusökning.

2 Bakgrund: Statoils kunder och kostnader

Statoil har sju stycken depåer som ska förse 237 stycken bensinstationer med bensin. Varje kunds efterfrågan liksom geografiska placering är känd. I figur 1 har Statoils kunder markerats med en svart prick och Statoils depåer markerats med en röd prick. En tankbilstyp används och tankbilen har en kapacitet på 48 kubikmeter. Logistikkostnaden delas upp i tre olika kostnadsposter.



Figur 1: Karta över Statoils depåer och kunder

- Det kostar 300 SEK att starta en rutt.
- Det kostar 40 SEK att lämna bensin hos en kund.
- Transportkostnad om 12 SEK/km.

Värt att notera är att totala kostnaden att lämna bensin på bensinstationerna kommer att vara konstant (9480 SEK), oberoende vilken rutt som används. Detta beror på att alla bensinstationers efterfrågan måste uppfyllas och att en bensinstation bara kan besökas en gång (delleveranser tillåts alltså inte). En annan begränsning i problemet är kapaciteten hos tankbilarna, vi har utgått från att Statoil kommer att använda sig av tankbilar som klara att lasta 48 kubikmeter bensin.

3 Tilldela kunder till depåer

Att tilldela ett visst antal kunder till en depå är ett såkallat tilldelningsproblem (assignment problem) och kan i vårt fall formuleras på följande sätt:

$x_{ij} = 1$ om kund i tilldelas en depå j, 0 annars.
Låt c_{ij} vara avståndet mellan kund i och depå j, d_i vara efterfrågan för kund i och cap_j vara kapaciteten för depå j.

$$\min z = \sum_i \sum_j x_{ij} \cdot c_{ij} \tag{1}$$

då

$$\sum_i d_i \cdot x_{ij} \leq cap_j \qquad \forall j \tag{2}$$

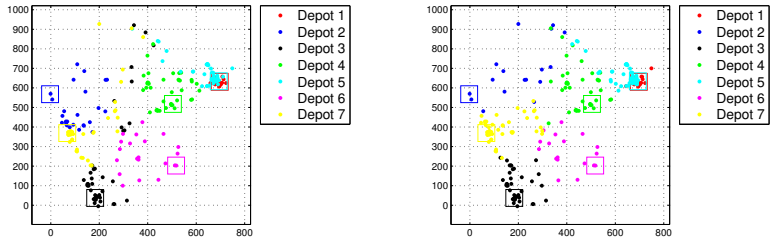
$$\sum_j x_{ij} = 1 \qquad \forall i \tag{3}$$

Det är ett LP-problem som inte torde ta särskilt lång tid att lösa, även för ett stort problem. Vi valde emellertid att använda en enkel heuristik som implementerades i Matlab för att lösa problemet. Anledningen till det är att vi för enkelhetens skulle ville använda samma mjukvara till hela uppgiften. Men vi har också testat att lösa problemet med CPLEX i AMPL, det ger oss en optimal lösning till vårt tilldelningsproblem.

I vår heuristik blev vi inspirerade av Prim's algoritm och valde att tilldela kunder till en depå genom att titta på den kund som har kortast avstånd till en depå och försöka tilldela kunden till depån. Vår enkla heuristik innehåller följande steg:

1. Steg 1: Hitta den kund som ligger närmast en depå.
2. Steg 2: Kolla om det finns tillräckligt med kapacitet hos depån för att täcka kundens behov.
3. Steg 3: Om ja i steg 2, tilldela kunden till depån. Om nej i steg 2, gå vidare till nästa depå och börja om på steg 1.
4. Algoritmen körs tills alla kunder har tilldelats en depå.

Vi har också löst det optimala tilldelningsproblemet. De olika tilldelningarna illustreras i figur 2.



(a) Tilldelning med vår heuristik (b) Optimal tilldelning

Figur 2: Resultat av olika tilldelningsmetoder

4 Undre gräns för kostnad av systemet

För att utvärdera resultatet är det vettigt att räkna ut en optimistisk skattning, det vill säga en kostnad som vi med säkerhet vet ligger under den minsta kostnaden för hela systemet. Kostnaderna som är kopplade till systemet finns redogjorda för i avsnitt 2.

Den totala kostnaden för att stanna hos bensinstationerna kommer att vara samma oavsett vilka rutter som körs. Den kostanden uppgår till 9 480 SEK. Den ackumulerade efterfrågan för alla bensinstationer är 4309 kubikmeter och eftersom att en tankbil har kapacitet på 48 kubikmeter kommer det som minst att krävs $\lceil \frac{4309}{48} \rceil = 90$ antalet rutter. Att starta en rutt kostar 300 SEK, det vill säga det kommer som minst att kosta $300 \cdot 90 = 27000$ SEK att starta rutterna.

För transportkostnader blir det svårare att göra en skattning. Vi har bestämt att göra beräkningen på följande sätt:

Det kortaste avståndet mellan en depå och en bensinstation är tre kilometer.

Det kortaste avståndet mellan två bensinstationer är tre kilometer.

I snitt kommer $2,6$ bensinstationer att besökas per rutt. Transportkostnaden för en rutt beräknas nu enligt följande: $(3 \cdot 2 + 1,6 \cdot 3)12 = 130$ SEK Transportkostnaden för 90 av samma rutt blir således 11700 SEK. Summerat kommer den undre kostnaden för systemet att bli $9480 + 27000 + 11700 = 48180$ SEK. Notera att detta är en mycket optimistisk skattning. Särskilt är transportkostnaden (som drivs av längden på sträckan) lågt skattad och kommer i praktiken att bli högre. Skattningen bör således användas med stor försiktighet eftersom att transportkostanden är den största kostnadsdrivaren i systemet.

5 Tabusökning

5.1 Vad är tabusökning?

För att hitta en optimal lösning till problemet krävs otrolig datorkraft och/eller tid. Därför behövs en heuristik som systematiskt kan iterera över olika möjliga lösningar för att vi ska kunna hitta en god lösning. För att bestämma vilka rutter som tankbilarna ska köra så använde vi oss utav tabusökning. Tabusökning är en sorts heuristik där man har som mål att undvika lokala optimum, eller ta sig ur dem (Tabu Search: A Tutorial, 2001). Genom att börja i en startlösning och sedan modifiera den systematiskt i varje iteration, och göra den bästa modifieringen till "tabu", vilket innebär att man gör det otillåtet för heuristiken att göra samma modifiering igen inom ett visst antal iterationer, så kan man hitta nya lösningar till problemet och på så sätt leta sig ur lokala optimum.

Efter att varje bensinstation (som vi ibland väljer att kalla kunder) tilldelats en depå, så är problemet som finns kvar ett så kallat Vehicle Routing Problem (VRP). Problemet är således att hitta rutter för tankbilarna som minimerar den totala kostnaden för systemet.

Det kommer alltså att finnas ett VRP för varje depå. Vi har valt att lösa varje sådant problem separat och sedan "slå ihop" denna lösning till en lösning som löser hela problemet (det vill säga att hitta en så bra ruttplanering som möjligt för alla depåer). Nedan beskrivs hur vår tabusökning fungerar för *en* depå.

Tabusökningen är till en början ganska simpel, men den har efterhand utökats och gjorts mer avancerad. I sin grund ser dock tabusökningen ut enligt följande:

1. Steg 1:
   Skapa en tillåten startlösning (se avsnitt 5). Men vår startlösning kommer att innehålla en lista över alla kunder som tillhör en depå, viktigt att notera är att ordningen som kunderna kommer att ligga är i är av intresse.

2. Steg 2:
   I detta steg gör vi vår första modifiering av vår nuvarande lista över bensinstationer. Genom att ändra på ordningen i vår nuvarande lista, så kan vi sedan hitta nya lösningar. Ordningen ändras genom att vi byter plats på två närliggande bensinstationer i vår lista.

3. Steg 3:
   Använd den nya ordningen för att skapa rutter för tankbilarna. En rutt skapas genom att vi lägger till den första kunden (bensinstationen) i en rutt. Om det finns plats för att lägga till ännu en kund, det vill säga nästa kund i listan, så läggs även den till i rutten. Processen fortsätter tills det att det inte finns mer kapacitet i tankbilen för att den ska kunna uppfylla efterfrågan hos nästa kund i listan. Då börjar vi om med en ny rutt och den första kunden, som inte har blivit tilldelad en rutt, i listan läggs till nästa rutt. Man kan säga att vi försöker skapa så få rutter som möjligt, eftersom det är dyrt att ha många rutter.

4. Steg 4:
   Steg 3 körs för alla byten av två närliggande kunder som gjordes i steg 2, det byte som gav upphov till bäst ordning kommer att ge den lägsta ruttkostnaden i steg 3. Vi jämför alla byten med varandra och väljer att gå vidare med det byte som gav upphov till lägst ruttkostnad och som inte är tabu.

5. Steg 5:
   Tabulistan uppdateras och det blir tabu att göra samma byte.

6. Steg 6:
   Börja om på steg 2.

5.2 Exempel på hur tabusökning fungerar

Vi har förstått att det var ett ett tag sedan ni läste optimeringslära och att förklaringen ovan därför kan uppfattas som abstrakt. Därför vill vi i detta avsnitt ge ett kort exempel på hur vår tabu fungerar:

1. Steg 1:
   Säg att vi har en lösning med tillåtna rutter och säg att ordningen på kunderna i listan över kunderna är: [1,2,3,4,5].

2. Steg 2:
   Modifiera listan över kunder. Modifieringen kommer ge följande lösning:

   (a) $[2,1,3,4,5]$
   (b) $[1,3,2,4,5]$
   (c) $[1,2,4,3,5]$
   (d) $[1,2,3,5,4]$
   (e) $[5,2,3,4,1]$

3. Steg 3:
   Hitta rutter. Nu kommer vi använda våra modifierade listor, från steg 2 för att skapa tillåtna rutter. Det skulle till exempel kunna se ut som följer: (bensinstationerna inom en hakparentes utgör en rutt).

   (a) $[2,1][3,4][5]$
   (b) $[1,3,2][4][5]$
   (c) $[1,2][4,3][5]$
   (d) $[1,2,3,5][4]$
   (e) $[5,2][3,4][5]$

   Det som är intressant att se här är att det blir olika rutter beroende på ordningen, det sker på grund av att kunderna har olika efterfrågan och då kommer det få plats olika många kunder i en rutt beroende på hur de är ordnade i listan.
   Vi räknar nu ut kostnaden för de olika rutterna:

(a) 5000kr

(b) 5800kr

(c) 5200kr

(d) 4800kr

(e) 5900kr

4. Steg 4:
   Vi ser tydligt att rutt nr 4, har lägst kostnad, alltså den med ordningen i vår kundlista: [1,2,3,5,4]. Detta blir vår nya ordning!

5. Steg 5:
   Vårt tabu i detta fall blir att byta på kunderna: 5,4 (däremot kan 4 byta plats med kund 1 och 5 kan byta plats med kund 3.).

6. Nu har vi gått igenom en iteration och vi kommer börja om från Steg 2, med den nya ordningen som gavs i steg 4.

Vår tabusökning kommer alltså att göra lika många många byten som det finns kunder, minus antalet kunder som finns i tabu. Det gör att vi har ett ganska stort område att söka igenom, men det går relativt snabbt att köra tabusökningen.

6 Hur en tillåten lösning hittas

Det finns flera sätt att hitta en tillåten lösning på. En väldigt enkel metod är den som används i vår tabusökning, det vill säga lista alla kunder som hör till en depå och låt första rutten bli de kunderna, i den ordningen de står listade, sålänge kundernas efterfrågan är lägre än tankbilens kapacitet. De övriga rutterna skapas på samma sätt tills alla kunder ingår i en rutt. Vi vill dock ha en relativt bra startlösning, främst av den anledningen så att vi kan utvärdera hur bra vår tabusökningen fungerar. Följande heuristik används:

Steg 1: Hitta den bensinstation som ligger närmast en depå. Lägg till bensinstationen i rutten. Ta bort från listan av kandidater.

Steg 2: Lägg till de bensinstationer som ligger närmast *och* får plats i rutten. Ta bort de bensinstationer som läggs till från listan av kandidater.

Steg 3: Börja om på steg 1 tills alla bensinstationer blivit tilldelad en rutt.

7 Resultat från tabusökning

7.1 tabuSimple

Vi vill presentera en del resultat från vår tabu, som den beskrivs ovan, denna tabu kallas här för Simpel Tabu.

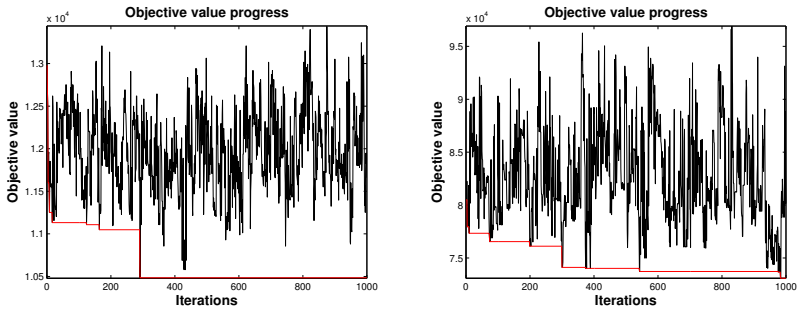
Vi har valt att implementera vår tabu med följande stoppkriterium, den stoppas efter att det har körts 1000 “iterationer per depå”, där en iteration definieras som att vi har gjort steg 2-5 (I praktiken sker alltså fler iterationer, då steg 2 (modifieringen) också innehåller ett antal iterationer som beror av längden på kundlistan och tabulistans längd).

Vår Tabu-lista kan max innehålla 10 element och ett element kommer att ligga kvar i tabulistan i 10 iterationer, det betyder att en modifiering, inte kan ske igen på 10 iterationer.

Med dessa förutsättningar beror målfunktionsvärdet på vilken bensinstationstilldelning vi använder. Vi testade med båda och det visade sig att reslutatet skiljer sig mycket. Med tilldelningen enligt den heuristiken som redogörs för i avsnitt 3 fås kostnaden 450 444 SEK och med tilldelning enligt den matematiska modell det redogörs för i samma avsnitt får vi en kostnad på 381 648 SEK. Vi testade med olika tabulängd och fick för samtliga fall att den matematiska modellen presterade ungefär 15 % bättre, därför har vi valt att använda den tilldelningen för vidare analys. 381 648 SEK kan jämföras med vår tidigare lösning som vi fick innan vi använde vår tabusökning (beskrivs i avsnitt 6), där vårt målfunktionsvärde var 529 308 SEK. Det är imponenerande hur en så pass enkel algoritm snabbt (ungefär en minut) kan hitta en mycket bättre lösning till ett så komplext system. Denna tabusökning har alltså lett till en kostandsbesparing på 147 660 SEK (en kostnadsminskning med ungefär

28 %). Nedan redgörs kostnadsförändringen för depå 1 respektive depå 2. Den röda linjen indikerar lägsta kostnaden fram tills den iterationen och svarta linjen indikerar kostnaden vid varje iteration. Särskilt intressant är hur depå 2 succesivt hittar bättre lösningar, vilket indikerar att det kan vara värt att köra många iterationer. Grafer för övriga depåer har ungefär samma utseende, men hittar sitt lägsta värde något snabbare.

Vänligen kör MATLAB-koden som finns bifogad till denna rapport för samtliga grafer.



(a) Resultat från tabuSimple för depå 1 (b) Resultat från tabuSimple för depå 2

Figur 3: Delar av resultatet från tabuSimple

Tabulistan skulle kunna göras mer avanacerad, exempelvis genom att justera dess längd beroende på hur många bensinstationer som är tilldelad till respektive depå. Vi har valt tabulistans längd till 10, det fungerar bra för de depåer som har ett mindre antal kunder, men vissa depåer har ett betydligt större antal kunder, t.ex. depå 7 har 53 kunder, vilket gör att det skulle kunna vara bra att ha en längre tabulista för denna depån. Därför kan det vara en god idé att undersöka om en justerad längd på tabulistan kan förbättra resultatet.

7.2 Diversifiering

Diversifiering kan vara bra att använda sig av för att hitta nya områden att söka av. Två modifieringar av tabusökningen har gjorts, en som kallas tabuSimpleDiversification och en som kallas tabuAdvancedDiversification. Tanken bakom båda är att på olika sätt diversifiera sökningsområdet. Med det menas att skapa möjligheten att lämna ett "område", för att kunna hitta nya bra lösningar, som inte ligger i området som vi kan ha "fastnat" i.

7.2.1 tabuSimpleDiversification

I tabuSimpleDiversification så räknas antalet iterationer sedan ett nytt bästa värde hittades. Om det har gått många iterationer från det att vi hittade ett nytt bästa värde, finns risken att det kommer ta väldigt lång tid innan en bättre lösningen hittas. Vi har då valt att tömma vår tabulista och slumpa fram en ny startlösning. Från våra tester så

verkar det som att 50 iterationer från det att vi hittade en ny bästa lösning är en lämplig väntetid, men i framtiden kan det vara bra att testa med andra värden på väntetiden (50 iterationer funkar bra om det totala antalet iterationer är 1000 stycken). Eftersom att en ny lösning slumpas fram ökas antalet iterationer till 5000. Denna funktionalitet implementerades och totala kostnaden för systemet blir 380088 SEK, vilket är lite mindre än vad vi fick i tabuSimple.

7.2.2 tabuAdvancedDiversification

Låt oss se hur lösningen blir om vi använder oss av tabuAdvancedDiversification. I tabuAdvancedDiversification har vi använt oss av tabuSimpleDiversification men vi har gjort ett tillägg, om den bästa lösningen som hittas i en iteration är sämre än den föregående lösningen och om det har sett ut så i flera iterationer så stannar vi och hittar ett nytt startområde för vår tabusökning, samt tömmer tabulistan. Detta för att vi kan anta att vi för ett antal iterationer sedan har gjort en dålig modifiering och att vi nu börjar röra oss mot ett dåligt område, vi vill snabbt avbryta det och komma tillbaka till ett bra område igen.

Man kan variera hur många iterationer som man vill tillåta att man ska kunna få ett sämre målfunktionsvärde, innan vi går till ett nytt startområde. Låt parametern x vara (i koden heter parametern advancedTabuLength) antalet iterationer som vi tillåter ett efterföljande sämre målfunktionsvärde. Vi har experimenterat med x mellan 3 och 10, det visar sig att för x större än 6, så fås inga användningar av advancedDiversification, vid ett så stort x så fungerar koden på samma sätt som för tabuSimpleDiversification. Man kan på värdet hos parametern "timesUsedAdvancedTabu" se hur många gånger som advancedDiversification aktiveras, för olika x värden ser det ut som följer:

1. x = 3 ger 199 användningar av advancedDiversification,
   målfunktionsvärde = 383 748 SEK
2. x = 4 ger 52 användningar av advancedDiversification,
   målfunktionsvärde = 384 180 SEK
3. x = 5 ger 14 användningar av advancedDiversification,
   målfunktionsvärde = 383 532 SEK
4. x = 6 ger 5 användningar av advancedDiversification,
   målfunktionsvärde = 383 532 SEK
5. x = 7 ger 0 användningar av advancedDiversification,
   målfunktionsvärde = 383 628 SEK (samma som i SimpleDiversification (om båda körs för 1000 iterationer)).

Tyvärr ger tabuAdvancedDiversification inte bättre lösningar än tabuSimpleDiversification. Det kan vara så att det är bra att tillåta vår tabusökning att fortsätta att köra längre än vad advancedDiversification tillåter, det kanske är nödvändigt för att man ska kunna komma ut ur lokala optimum. AdvancedDiversification begränsar hur långt man

kommer från en lösning, om efterföljande lösningar ger sämre målfunktionsvärden. För att hitta ännu bättre lösningar kanske man kan använda sig av någon annan diversifierings- eller intensifieringsteknik.

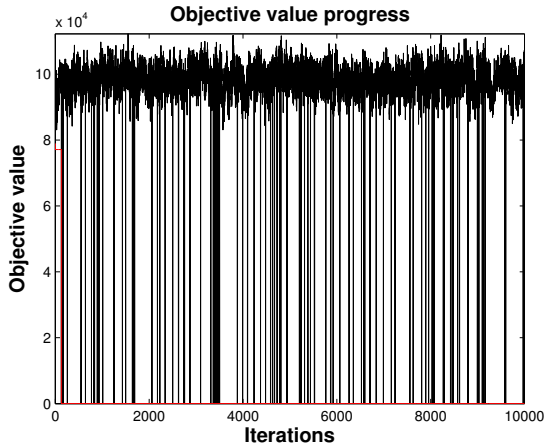
7.2.3 tabuInfeasible

En annan idé för att hitta nya områden i tabusökningen är att tillfälligt tillåta otillåtna lösningar under tiden som man söker efter nya värden. Vi har implementerat detta i matlab och koden finns i tabuInfeasible.m. tabuInfeasible bygger på tabuSimple, men den skapar otillåtna lösningar genom att den skapar två rutter, en vanlig rutt skapas på samma sätt som i tabuSimple, men den skapar även en otillåten rutt där kapaciteten på tankbilen har höjts till 60. Den otillåtna lösningen straffas och straffet är olika stort beroende på hur länge den otillåtna lösningen har varit bättre än den tillåtna, med bättre menar vi att den otillåtna lösningen har lägre ruttkostnad (trots straff). Straffet räknas ut på följande sätt: penalty = timesInfeasible·500 + 2000

Där timesInfeasible är antalet gånger i rad som en otillåten lösning har varit bättre än en tillåten. Ett fast straff har satts till 2000.

tabuInfeasible ger efter 10 000 iterationer ett målfunktionsvärde på: 385 740 SEK.
Om man "bara" kör 1000 iterationer, vilket är det antal iterationer som vi har valt i de andra tabusökningarna, så fås ett målfunktionsvärde på: 390 612 SEK.

Från resultatet på vår tabuInfeasible kan vi se att den hittar nya lösningar. Man kan se det både i graferna som skapas från körningen av tabusökningen (som bifogas till rapporten) men också på vårt målfunktionsvärde. Det är visserligen inte lika bra som varken "tabuSimple", "tabuSimpleDiversification" eller "tabuAdvancedDiversification", men kanske skulle det bli bättre om man la till fler faktorer till tabuInfeasible, så som liknande diversifiering som sker i "tabuSimpleDiversification". Man kan också anta att om man ställer in straffparametrarna och även den otillåtna kapacitetsnivån i skapandet av otillåtna rutter, så skulle man kunna få en bättre tabusökning som använder sig av otillåtna rutter. Det är något vi rekommenderar er att kolla vidare på.



Figur 4: Exempel på lösning från tabuInfeasible för Depå 4

Värt att notera är att otillåtna lösningar inte tillåts existera i slutlösningen. Då det inte skulle ge några rutter som ni skulle kunna använda er av. De otillåtna lösningarna används alltså bara för att hitta nya områden, men inte i själva resultatet från sökningen. Figur 4 är ett visar på hur “tabuInfeasible” har hittat lösningar för hur ruttplaneringen för depå 4 skulle kunna se ut. I figuren så visas alla lösningar som tabusökningen hittas, men alla dessa används inte i lösningen som “tabuInfeasible” ger.

8 Resultat och diskussion av resultat

Resultatet visar att den enklaste algoritmen vi byggde fick det bästa målfunktionsvärdet. Att de mer avancerade tabusökningarna inte fick ett bättre resultat tror vi beror på följande tre anledningar:

1. Antalet iterationer vi körde är rätt få. Problemet är stort och det förefaller därför troligt att många iterationer krävs för att hitta en bättre lösning. Med mer tid, datorkraft och bättre implementerad programkod skulle antalet iterationer kunna öka väsentligt. Då är det också troligt att bättre lösningar skulle kunna ha hittats.

2. Vi har inte i högre utsträckning justerat paramatervalen tillräckligt mycket, t.ex. tabulistans längd. För "tabuSimpleDiversification" och "tabuAdvancedDiversification" så borde man också försöka justera in dess parametrar till bättre värden för att hitta ett bättre resultat.

3. Vi har ett ganska stort sökområde (neighbourhood) till vår tabusökning. Det gör att den tar längre tid att köra men den hittar också bra lösningar, utan att man behöver diversifiera området. Om man då ändå väljer att göra en diversifiering så begränsar man tabusökningens område, i varje fall med de metoder som vi använde oss av, vilket gör att resultatet intte kommer bli lika bra.

Oavsett så har vi lyckats hitta en fullgod lösning till problemet som minskat kostnaderna med 28 % jämfört med vår initiala heuristik. Ponera att efterfrågan på bensin gäller per dag, då skulle detta innebära en kostnadsminskning på 53 MSEK årligen.

För er ruttplanering så rekommenderar vi er att använda er av tabuSimple. Som vi nämnt ovan så ger den målfunktionsvärdet: 379 320 SEK (10 000 iterationer), vilket är en förbättring på 149 988 SEK SEK eller 28,34% (jämfört med lösningen från den första tillåtna lösningen som gav målfunktionsvärde: 529 308 SEK.).

Det kan vara värt att undersöka om Statoil har möjligheten att hyra/köpa lastbilar som har högre kapacitet. Vid högre kapacitet kommer kostnaden att minska på grund av att antalet rutter kan minska, vilket också kommer att minska den totala körsträckan. Å andrasidan kan kilometerkostnaden öka något, eftersom att energiförbrukningen bör vara högre för ett stort fordon, men den kostnaden torde vara marginell i sammanhanget (Statoil lär ju ha tillgång till billigt drivmedel!). Med kapaciteten 60 kubikmeter får vi den lägsta totalkostnaden: 326 880 SEK, vilket betyder att den har en kostnadreducering på 14% jämfört med vår bästa funna lösning. Finns möjligheten att hyra större lastbilar (60 kubikmeter) för en kostnad som är lägre än 54 000 SEK per dygn, bör man överväga det. Det skulle förutom de ekonomiska få andra positiva effekter, bland annat mindre miljöpåverkan eftersom att den totala körsträckan skulle minska. Å andra sidan bör man se upp så att lastbilschaufförerna inte får för lång körsträcka. Er personalhälsa är viktig!

För de olika depåerna fås följande data:

Tabell 1: Data rutt

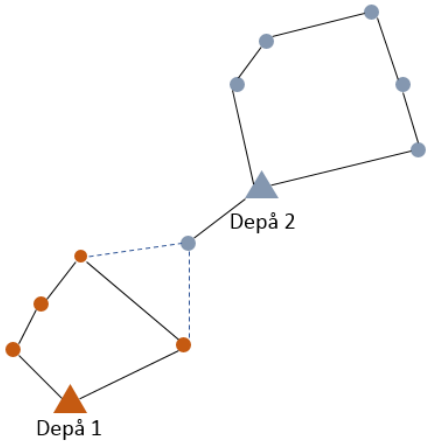
	Depå 1	Depå 2	Depå 3	Depå 4	Depå 5	Depå 6	Depå 7	Totalt
Antal rutter	7	10	20	19	22	9	22	109
Antal kunder	17	17	41	42	49	18	53	237
Min kunder per rutt	1	1	1	1	1	1	1	1
Max kunder per rutt	4	3	4	4	4	3	4	4
Kortaste rutt [km]	26	10	12	25	32	14	18	10
Längsta rutt [km]	216	1062	579	893	685	628	690	1062
Genomsnittlig rutt [km]	91	578	198	318	181	356	206	260

I genomsnitt så besöks 2.1944 bensinstationer per rutt.

I snitt kostar en rutt 3512 kr.

Vi valde initialt att tilldela bensinstationer en depå, genom två metoder. Det är viktigt att poängtera att dock inte att denna lösningen inte nödvändigtvis kommer att vara optimal i det stora problemet, där hänsyn tas till både ruttplanering och tilldelning. Det kan mycket väl vara så att även om tilldelningen är optimal med avseende på avstånd så finns det andra faktorer som påverkar den “riktiga” optimal lösningen (till stora problemet), exempelvis hur många rutter som det blir, vilket beror på tankbilarnas kapacitet och kundernas efterfrågan. En annan faktor är vilka avstånd som fås mellan kunder i den optimala ruttlösningen.

Denna problematik kan lätt exemplifieras genom systemet i Figur 5, där tilldelningen gjorts enligt den matematiska modell som presenteras ovan. Ponera att en tankbil har kapaciteten 30 kubikmeter och att varje kund har efterfrågan 5 kubikmeter. Med den tilldelning hade en lösning varit enligt figur 5, men med de sträckade linjerna kan vi enkelt visa att det finns en bättre lösning om kunden som ligger sydväst om depå 2 tilldelads depå 1.



Figur 5: Fiktivt system

Slutsatsen vi kan dra är att om vi löser de sju subproblemen optimalt, är det inte säkert att vi hittat optimum globalt. Därför kan det vara en god idé att för framtiden utöka modellen till att också förändra tilldelningen. Men man bör dock undersöka/göra en bedömning hur mycket man skulle kunna tänka sig spara för att se om det är värt att investera den tid det skulle ta att implementera.

9 Allmäna reflektioner

Vi har reflekterade över två delar av projektet som var svårt.

1. Det var initialt svårt att komma på en bra tabusökning och hur den skulle implementeras. Det var också svårt att veta vilka tabulängd man skulle använda.

2. De andra uppgifterna var svåra på så sätt att det vara flera parametrar som kunde ändras och det var klurigt att veta vad man skulle ändra för ett bättre resultat.

Kul med implementation i Matlab, det känns som att det är mer realistiskt att man kommer använda ute i arbetslivet än AMPL. Det var bra att det fanns en del färdigskrivna program, bland annat visualisering över depå, rutter och kostnadsförändring. Kul projekt.

Referenser

[1] European commision, *Reducing CO2 emissions from vehicles*,
url: http://ec.europa.eu/clima/policies/transport/vehicles_en
hämtad 2016-12-04.

[2] University of Colorado, *Tabu Search: A Tutorial*,
url: http://www.ida.liu.se/ zebpe83/heuristic/papers/TS˙tutorial.pdf
hämtad 2016-12-05.