Linköpings Tekniska Högskola
Matematiska institutionen/Optimeringslära
Kaj Holmberg

2019–03–13

Laborationsinformation

TAOP88 Optimering för ingenjörer, VT19

1 Information om SNOWPLAN

SNOWPLAN version 3 är ett python-program som hjälper till vid lösning av ett förenklat snöröjningsproblem. (Version 1 och 2 var skrivna i matlab.) Problemet som löses kan kallas ett "k-brevbärarproblem", eftersom det är ett brevbärarproblem där alla bågar ska täckas (minst en gång), men inte av *en* brevbärare utan av flera (k st). En "brevbärare" motsvarar i detta sammanhang ett snöröjningsfordon.

SNOWPLAN arbetar med ett subproblem, som består av ett lantbrevbärarproblem ("rural postman problem") för varje fordon. Detta problem löses med en kod från Vineopt, *rupov*. (För att SNOWPLAN ska hitta den koden, krävs att man har kört Vineopt i den katalog där man ska köra SNOWPLAN.) Nätverket läses in i Vineopt-format, och huvudfunktionen hos SNOWPLAN är att finna en bra uppdelning av bågarna mellan de olika fordonen.

Två viktiga vektorer som programmet arbetar med är `order`, som är ordningen i vilken fordonen behandlas, och `vehdo`, som för varje båge anger vilket fordon som ska röja bågen.

Huvudprogrammet SNOWPLAN innehåller följande delar.

1. Nätverket läses in.
2. En startordning konstrueras.
3. En startallokering av bågar till fordon görs.
4. Lantbrevbärarproblemet löses för varje fordon.
5. Valfritt: De bågar ett fordon passerar anses röjda av fordonet, om bågen inte tidigare är röjd, oavsett om bågen var allokerad till att annat fordon.
6. Valfritt: Lösningen förbättras genom att flytta cykler från en tur till en annan.
7. Ordningen för fordonen ändras.
8. Allokeringen av bågar till fordon ändras.
9. Steg 4 - 9 upprepas ett visst antal gånger.
10. Allokeringen av fordon till bågar skrivs ut (som bågmängder), i ett format som Vineopt kan läsa in.

I varje iteration beräknas totala kostnaden som summan av kostnaderna för varje fordons tur plus de fasta kostnaderna för fordonen. Dessutom beräknas tiden för lösningen som den maximala av fordonens tider. Målfunktionsvärdet beräknas sedan som en viktad summa av kostnaden och tiden.

För att acceptera förflyttning till en ny punkt (för `vehdo`) används simulerad kylning, dvs. bättre punkter accepteras alltid och sämre punkter accepteras med en viss sannolikhet.

Punkterna 3, 7 och 8 kan göras på flera olika sätt, så där måste man välja alternativ, och man måste också välja vilka av de valfria punkterna 5 och 6 som ska göras. Dessa val styrs av parametrar som användaren sätter.

Observera att programmet använder *slump*, så två körningar med identiska inställningar kommer inte att ge precis samma resultat.

1.1 Start av programmet

Programmet startas genom att öppna ett terminalfönster och skriva `/courses/TAOP88/dine 6`.

Man får då upp den interaktiva sidan med möjligheter att läsa in nätverk och parametrar, samt ändra parametrar och lösa problemet. Man kan även ge namn på nätverksfil och parameterfil på kommandoraden:

```
/courses/TAOP88/dine 6 network1 param1.par
```

läser in nätverket *network1* och parameterfilen *param1.par*. Om man ger

```
/courses/TAOP88/dine 6 network1 param1.par 1
```

kommer problemet att lösas direkt. Man kommer då att få se hur målfunktionsvärdet iterativt förändras. En blå linje ger ursprungligt målfuktionsvärde i varje iteration, medan gröna kryss ger värdena efter förbättringsfasen, och den röda streckade kurvan ger det bästa värdet efter varje iteration. En ljusblå linje ger den undre gränsen.

Om man ger

```
/courses/TAOP88/dine 6 network1 param1.par 2
```

kommer problemet att lösas, varefter programmet avslutas. För att se hur det gick, får man då studera utskrifterna på terminalen.

Varje gång ett problem löses, adderas en rad till filen *fkres.txt*, med nätverksnamn, inparameterfil, datum, problemstorlek samt målfunktionsvärde och lösningstid. Denna raderas aldrig, så man rekommenderas byta namn på den då man gjort alla körningar som man vill jämföra. En liknande fil, *fpkres.txt*, skapas på samma sätt, men innehåller (kompakt) information om alla parametrar.

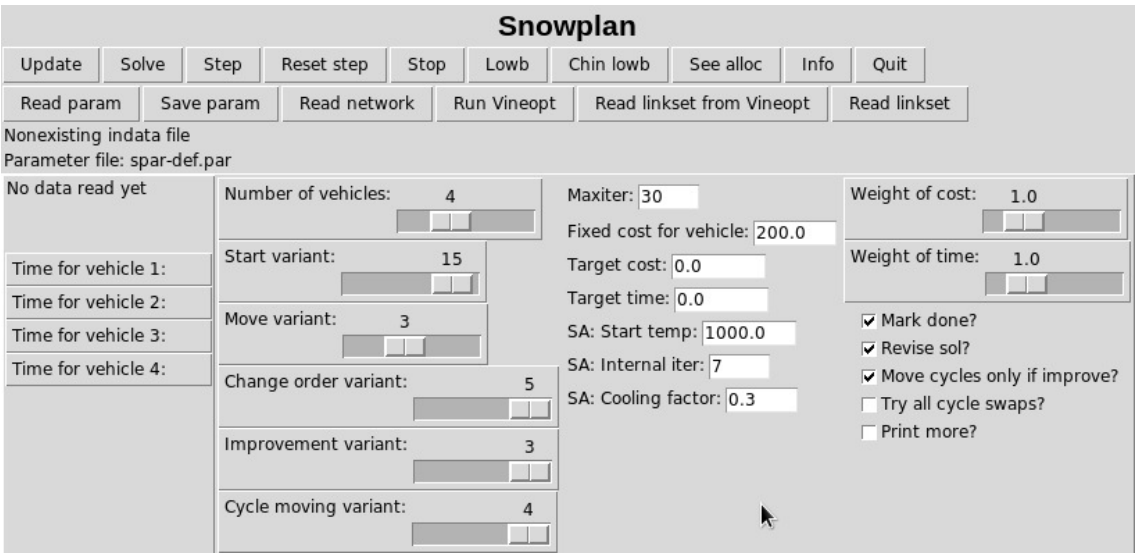
Dessutom skapas flera resultatfiler i datakatalogen (# står för ett nummer som räknas upp varje gång):
network1-param1_#.png: Bild av bästa allokeringen i png-format.

network1-param1_#.ps: Bild av bästa allokeringen i postscript-format.
network1-param1_#.val: Undre gräns(er) samt för varje iteration målfunktionsvärde före och efter förbättring, samt det bästa värdet dittills.
(Rensa gärna bland dessa filer ibland.)

1.2 Parametrar

Programmet använder flera parametrar som styr lösningsmetoden, och dessa kan läsas in från en fil. Ett exempel på förslagna värden ges i filen *spar-def.par*. Om man finner bättre värden på parametrarna, sparas dessa lämpligen på en annan fil (med extension *.par*).

Man kan också ställa in parametrarna i programmets interface, se bilden nedan.



När man har ändrat någon parameter, bör man klicka på knappen "Update" för att inställningarna säkert ska sparas. Om man glömt vad de olika värdena står för, kan man klicka på "Info". Man får även informativa utskrifter i terminalen, när man ändrar någon parameter.

För att beräkna en undre gräns, klicka på "Lowb" eller "Chin lowb".

När man klickar på "Solve" löses problemet. Om man vill göra en iteration i taget, ska man istället klicka på "Step".

Om man klickar på "Run Vineopt", startas Vineopt, och nuvarande allokering (bågmängder) läses in och visas. Man kan då ändra allokeringen, och spara den via menyn "Sets", "Save link sets for Snowplan". Därefter avslutar man lämpligtvis Vineopt, och klickar på "Read linkset from Vineopt" i Snowplan, varvid Snowplan läser in och utgår från allokeringen som sparats i Vineopt (om startvar=0). Man kan också sätta startvar=2, vilket gör att Vineopt-allokeringen läses in när man klickar på "Solve".

Knappen "See alloc" visar allokeringen i en färgad graf. (Denna funktion använder ett externt program, och kan fallera om programmet inte är installerat.)

1.3 Programmets funktion

SNOWPLAN finner först en tur för första fordonet som täcker alla bågarna som är allokerade till fordon 1. Om parametern *makedone*=1, markeras *alla* bågar som fordon 1 passerar som gjorda. M.a.o. följs inte allokeringen till fordon för dessa bågar. Det betyder att de följande fordonen får mindre att göra. Om man ändrar på ordningen i vilken fordonens turer bestäms, fås en annorlunda lösning.

Därför kan vektorn `order` ändras för att få olika lösningar. I detta projekt används flera olika möjligheter för att (slumpmässigt) ändra `order`.

På samma sätt finns flera olika möjligheter att ändra vektorn `vehdo`, för att förändra allokeringen av bågar till fordon.

Följande parametrar kan ges följande värden i parameterfilen. Om man inte ger något värde, får parameterna ett defaultvärde, som ges nedan inom parantes.

Parameter	Betydelse
nov	Antal fordon, ≥ 1 (4)
fixvehcost	Fast kostnad för ett fordon, ≥ 0 (200)
wcost	Vikt på kostnad i målfunktionen, ≥ 0 (1.0)
wtime	Vikt på tid i målfunktionen, ≥ 0 (1.0)
maxiter	Antal huvuditerationer, ≥ 1 (30)
logfil	Styr utskrifter från rupov (till skärmen eller fil), 0,1 (1)
pri	Mängd utskrifter (0 lite, 1 mycket), 0,1 (0)
markdone	Markera passerade bågar som gjorda, 0,1 (1)
startvar	Variant för startallokering, $0 - 16$, (15)
movevar	Variant för ändring av allokering, $0 - 7$ (3)
moalloc	Antal steg `vehdo` ska flyttas, ≥ 0 (1)
impvar	Variant för förbättringsheuristiken, $0 - 3$ (3)
dorevsol	Ändra allokeringen efter lösningen, 0,1 (1)
chordvar	Variant för ändring av ordning, $1 - 5$ (5)
moord	Antal steg `order` ska flyttas, ≥ 0 (1)
cymvar	Variant för flyttning av cykler, $1 - 4$ (4)
cymimp	Flytta cykler bara om det ger förbättring, 0,1 (0)
targetcost	Sluta när kostnaden understiger detta, ≥ 0 (0.0)
targettime	Sluta när tiden understiger detta, ≥ 0 (0.0)
starttemp	Starttemperatur i simulerad kylning, ≥ 0 (1000)
intiter	Antal iterationer med samma temperatur i simulerad kylning, ≥ 1 (7)
rcold	Kylningsfaktor i simulerad kylning, > 0, < 1 (0.3)

Filen *spar-def.par* kommer med dessa föreslagna värden på alla dessa parametrar. Vissa av parametrarna påverkar metodens prestanda mycket, andra mindre. Det är svårt att finna värden som är bäst för alla problem. Man kan behöva ändra vissa värden för att få metoden att fungera bra för vissa problem. Vissa värden kan ge bra lösningar, men gör att metoden tar för lång tid för stora problem. (Alltför små problem kan ha för få frihetsgrader för att optimering av parametrarna ske ge bra resultat.) För stora/svåra problem är det frestande att köra ganska få iterationer, men denna typ av metod ger då ingen bra lösning. Svåra problem tar lång tid att lösa, så man får vara tålmodig.

Efter varje körning sparas bågmängderna på filen *linksets#.set*, där # står för ett nummer

som räknas upp varje gång.

1.3.1 startalloc

Funktionen skapar en fordonsallokering, dvs. vektorn *vehdo*, där *vehdo(j)* = det fordon som ska fixa båge j.

Det styrs av parametern *startvar*:
startvar = 0: Starta från nuvarande lösning.
startvar = 1: Skapa en slumpmässig allokering.
startvar = 2: Läs bågmängder som sparats i Vineopt och använd som startallokering.
startvar = 3: Låt fordon 1 göra allt.
startvar = 4: Sortera i x för startnod, börja med första fordonet, fyll på närmaste, övergå sedan till nästa fordon.
startvar = 5: Som 4, men sortera i x för slutnod.
startvar = 6: Som 4, men sortera i x för mittpunkt.
startvar = 7: Som 4, men sortera i y för startnod.
startvar = 8: Som 4, men sortera i y för slutnod.
startvar = 9: Som 4, men sortera i y för mittpunkt.
startvar = 10: Som 4, men sortera i avstånd till origo för startnod.
startvar = 11: Som 10, men sortera i avstånd för slutnod.
startvar = 12: Som 10, men sortera i avstånd för mittpunkt.
startvar = 13: Använd koden *kmeans* som skapar ett antal kluster (lika många som antal fordon), där de närmaste noderna till varje klusters mittpunkt kopplas till klustret. Bågar med båda noderna i ett kluster allokeras till motsvarande fordon. Om ändnoderna tillhör olika kluster, allokeras bågen till klustret med högst nummer.
startvar = 14: Som 13, men om ändnoderna tillhör olika kluster, allokeras bågen till klustret med lägst nummer.
startvar = 15: Som 13, men om ändnoderna tillhör olika kluster, allokeras bågen slumpvis till ett av klustren.
startvar = 16: Som 13, men om ändnoderna tillhör olika kluster, allokeras bågen så att avståndet till centret av klustret minimeras.

Om man använder *startvar* = 0, kan man göra flera körningar efter varandra för att fortsätta förbättra lösningen. Att köra tre gånger med 30 iterationer ger nästan samma effekt som att köra en gång med 90 iterationer. Skillnaden är att temperaturen i simulerad kylning återställs efter varje körning.

1.3.2 changealloc

Funktionen ändrar allokeringen av bågar till fordon (vektorn *vehdo*).

Styrs av parametrarna *movevar* och *moalloc*:
movevar = 0: Ändra inte allokeringen.
movevar = 1: Byt två slumpmässiga länkar.
movevar = 2: Ändra *moalloc* slumpmässiga länkar.
movevar = 3: Flytta en från ett fordon som har mest till ett som har minst.
movevar = 4: Flytta en båge mellan två slumpade fordon, båda noderna gemensamma.

movevar = 5: Byt mellan två slumpade fordon, båda noderna gemensamma.
movevar = 6: Flytta en båge, en nod gemensam.
movevar = 7: Byt en båge, en nod gemensam.

1.3.3 changeorder

Funktionen ändrar ordningen. Styrs av parametrar *chordvar* och *moord*:
chordvar = 1: Byt två slumpmässiga platser i vektorn `order`.
chordvar = 2: Skapa helt ny slumpmässig ordning.
chordvar = 3: Skifta vektorn `order` *moord* steg åt höger.
chordvar = 4: Skifta vektorn `order` *moord* steg åt vänster.
chordvar = 5: Byt den som har flest mot den som har minst.

1.3.4 improvesol

Funktionen förbättrar lösningen på följande sätt. Finn en cykel i en tur, kolla om cykeln innehåller en nod i en annan tur, flytta cykeln till den andra turen.

Styrs av parametern *impvar*:
impvar = 0: Flytta inga cykler.
impvar = 1: Flytta cykler, ett slumpförsök.
impvar = 2: Flytta cykler, sluta när ingen förbättring sker.
impvar = 3: Flytta cykler, kolla alla kombinationer.

1.3.5 cyclemove

Körs om *impvar* > 0. Anropas från *improvesol*. Gör själva flytten av cykeln.

Styrs av parametern *cymvar*:
cymvar = 0: Gör inget.
cymvar = 1: Försök flytta en slumpad cykel mellan två slumpade fordon.
cymvar = 2: Försök flytta en cykel från en lång tur till en kort.
cymvar = 3: Försök flytta en cykel mellan givna fordon (för *impvar* = 3).
cymvar = 4: Försök flytta en cykel från en dyr tur till en billig.

1.4 Kontakt

Om, mot förmodan, programmet skulle krångla på något sätt eller inte fungera, ombedes användaren skicka information som gör att felet kan återskapas till kaj.holmberg@liu.se. Skicka då med parameterfilen med de inställningar som gör att programmet fallerar, samt information om vilket problem som körts. Man kan gärna ta en skärmbild av felutskrifter och skicka med.