Santa's Reindeer Challenge - A linear programming approach based on ZIMPL and LPSolve.

Rob Parker, Dec 2017

Problem

The DMCommunity.org has created a Christmas challenge. The challenge is replicated below;

Santa always leaves plans for his elves to determine the order in which the reindeer will pull his sleigh. This year, for the European leg of his journey, his elves are working to the following schedule, which will form a single line of nine reindeer. Here are the rules:

1. *Comet behind Rudolph, Prancer and Cupid*
2. *Blitzen behind Cupid*
3. *Blitzen in front of Donder, Vixen and Dancer*
4. *Cupid in front of Comet, Blitzen and Vixen*
5. *Donder behind Vixen, Dasher and Prancer*
6. *Rudolph behind Prancer*
7. *Rudolph in front of Donder, Dancer and Dasher*
8. *Vixen in front of Dancer and Comet*
9. *Dancer behind Donder, Rudolph and Blitzen*
10. *Prancer in front of Cupid, Donder and Blitzen*
11. *Dasher behind Prancer*
12. *Dasher in front of Vixen, Dancer and Blitzen*
13. *Donder behind Comet and Cupid*
14. *Cupid in front of Rudolph and Dancer*
15. *Vixen behind Rudolph, Prancer and Dasher.*

Approach

In this approach, I have used linear programming (LP) to solve the problem. To model the problem, I have used the ZIMPL modelling language. The problem is a constraint satisfaction rather than a minimisation or maximisation problem. Hence I have framed the problem as a minimisation problem, as this effectively constrains the solution space to the smallest set of ordinal numbers. I could have modelled the problem using integer programming methods, however the nature of the problem enables use of linear programming techniques where the output variables are integral anyway. Hence the problem is readily solved using linear programming solvers.

Model

I've modelled the order of reindeer as a sequential set of integers from 0 to 8. Hence if 0 represents the lead reindeer and 8 represents the last reindeer, the constraints can be represented as;

A in front of B, implies A<B

C behind D and E, implies C>D and C>E

Hence the ZIMPL model formulation is shown below;

```
# The set of reindeer
set REINDEER := {"Comet", "Rudolph", "Prancer", "Cupid", "Blitzen", "Donder", "Vixen", "Dancer",
"Dasher"};
# The decision variables where z represents the order or each reindeer
var z[REINDEER] real<= 8.0;

# Arbitrary objective function
minimize dummy : sum <r> in REINDEER : z[r];

# Comet behind Rudolph, Prancer and Cupid
subto a:
forall <r> in {"Rudolph", "Prancer", "Cupid"} :
   z["Comet"] - z[r]  >= 1.0;

# Blitzen behind Cupid
subto b:
   z["Blitzen"] - z["Cupid"]  >= 1.0;

# Blitzen in front of Donder, Vixen and Dancer
subto c:
forall <r> in {"Donder", "Vixen", "Dancer"} :
    z[r] - z["Blitzen"] >= 1.0;

# Cupid in front of Comet, Blitzen and Vixen
subto d:
forall <r> in {"Comet", "Blitzen", "Vixen"} :
   z[r] - z["Cupid"] >= 1.0;

# Donder behind Vixen, Dasher and Prancer
subto e:
```

```
forall <r> in {"Vixen", "Dasher", "Prancer"} :
    z["Donder"] - z[r]  >= 1.0;

# Rudolph behind Prancer
subto f:
    z["Rudolph"] - z["Prancer"]  >= 1.0;

# Rudolph in front of Donder, Dancer and Dasher
subto g:
forall <r> in {"Donder", "Dancer", "Dasher"} :
    z[r] - z["Rudolph"] >= 1.0;

# Vixen in front of Dancer and Comet
subto h:
forall <r> in {"Dancer", "Comet"} :
    z[r] - z["Vixen"] >= 1.0;

# Dancer behind Donder, Rudolph and Blitzen
subto i:
forall <r> in {"Donder", "Rudolph", "Blitzen"} :
    z["Dancer"] - z[r]  >= 1.0;

# Prancer in front of Cupid, Donder and Blitzen
subto j:
forall <r> in {"Donder", "Blitzen", "Cupid"} :
    z[r] - z["Prancer"] >= 1.0;

# Dasher behind Prancer
subto k:
    z["Dasher"] - z["Prancer"]  >= 1.0;

# Dasher in front of Vixen, Dancer and Blitzen
subto l:
forall <r> in {"Vixen", "Dancer", "Blitzen"} :
```

```
    z[r] - z["Dasher"] >= 1.0;

# Donder behind Comet and Cupid
subto m:
forall <r> in {"Comet", "Cupid"} :
    z["Donder"] - z[r]  >= 1.0;

# Cupid in front of Rudolph and Dancer
subto n4:
forall <r> in {"Rudolph", "Dancer"} :
    z[r] - z["Cupid"] >= 1.0;

# Vixen behind Rudolph, Prancer and Dasher
subto o:
forall <r> in {"Rudolph", "Prancer", "Dasher"} :
    z["Vixen"] - z[r]  >= 1.0;
```

Solution

The problem is readily solved using any linear programming solver. The ZIMPL pre-processor can be used to generate an LP model formulation or a number of solvers include ZIMPL as a pre-processor. I used LPSolve with a built in ZIMPL processor to solve the problem. LPSolve solves the problem very efficiently after 11 iterations of the simplex algorithm. The solution is shown below;

```
Variables; result
; 36
z$Blitzen; 4
z$Comet; 6
z$Cupid; 1
z$Dancer; 8
z$Dasher; 3
z$Donder; 7
z$Prancer; 0
z$Rudolph; 2
z$Vixen; 5
```