Topic 6: Case Studies
(Version of 13th November 2020)

Pierre Flener and Gustav Björdal

Optimisation Group
Department of Information Technology
Uppsala University
Sweden

Course 1DL441: Combinatorial Optimisation and Constraint Programming,
whose part 1 is Course 1DL451: Modelling for Combinatorial Optimisation
Outline

1. Black-Hole Patience
2. Antenna Placement
3. Warehouse Location
4. Sport Scheduling
Outline

1. Black-Hole Patience
2. Antenna Placement
3. Warehouse Location
4. Sport Scheduling
Black-Hole Patience

Move all the cards into the black hole. A fan top card can be moved if it is one rank apart from the black-hole top card, independently of suit (♣, ♥,♦,♠); aces (A,1) and kings (K,13) are a rank apart.


The cards $c_1$ and $c_2$ are one rank apart if and only if

$$(c_1 \mod 13) - (c_2 \mod 13) \in \{-12,-1,1,12\}$$

Defining a help predicate and avoiding $\text{mod}$ on variables:

1. `predicate rankApart(var 1..52: c1, var 1..52: c2) =
2. let { array[1..52] of int: R = [i mod 13 | i in 1..52] } in
   R[c1] - R[c2] in {-12,-1,1,12};`

Avoiding implicit `element` constraints for better inference:

2. `table([c1,c2], [|1,2|1,13|...|1,52|2,1|...|52,40|52,51|]);`

Let us model “adjacent black-hole cards are a rank apart”.
Model: Decision Variables and Constraints

Move all the cards into the black hole. A fan top card can be moved if it is one rank apart from the black-hole top card, independently of suit (♠, ♦, ♥, ♣); aces (A,1) and kings (K,13) are a rank apart.

Let Card[p] denote the card at position p in the black hole:

3 constraint Card[1] = 1;  % the card at position 1 is A♠

4 constraint forall(p in 1..51)(rankApart(Card[p],Card[p+1]));

Let us model “black-hole cards respect the order in fans”:

5 constraint forall ("fan with card" c1 “on top of” c2 “on top of” c3)
6   (let { var 2..52: p1; var 2..52: p2; var 2..52: p3 } in
7 % constraint alldifferent(Card);  % implied by correct data!

or, equivalently, without implicit element constraints:

6   (value_precede_chain([c1,c2,c3],Card));

Let us now formulate that second constraint even better.
Model: Redundant Variables & Channelling

Let \( \text{Pos}[c] \) denote the position of card \( c \) in the black hole. The black-hole cards respect the order in the given fans:

5 \text{constraint Pos[1] = 1;} \ % \text{the position of card A♠ is 1}
6 \text{constraint forall ("fan with card" c1 "on top of" c2 "on top of" c3)}
7 \quad (\text{Pos}[c1] < \text{Pos}[c2] \land \text{Pos}[c2] < \text{Pos}[c3]);
8 \ % \text{constraint alldifferent(Pos); } \ % \text{implied by correct data!}

How to model “adjacent black-hole cards are a rank apart” with the \( \text{Pos}[c] \) variables?!! Let us use the latter together with the \( \text{Card}[p] \) variables, and channel between them. Observe that \( \forall c,p \in 1..52 : \text{Card}[p]=c \iff \text{Pos}[c]=p. \)

Seen as functions, \( \text{Card} \) and \( \text{Pos} \) are each other’s inverse:

8 \text{constraint inverse(Card,Pos);}
\quad % \text{logically implies alldifferent(Card)}/\text{alldifferent(Pos)}

The model with mutually redundant variables and the 2-way channelling constraint is much faster (at least on a CP or LCG solver) than the models with only the \( \text{Card} \) variables.
Outline

1. Black-Hole Patience
2. Antenna Placement
3. Warehouse Location
4. Sport Scheduling
Antenna Placement Problem

Given:
- a region divided into zones, each with an expected gain if covered,
- the maintenance costs and covering areas of antenna types,
- a targeted number of antennae,

find: non-overlapping antenna placements and types such that the total expected gain of actual coverage minus the total maintenance cost is maximal.

We now show that we can pre-compute a 3d array with the net gain for each possible antenna placement and type, making it much easier to express the objective function and much faster to solve problem instances.
Model without Pre-Computation

... % z = #zones/dimension, t = #types, antennae = #antennae
2 set of int: Zones = 1..z; % the region has z by z zones
3 array[Zones,Zones] of int: ExpGain =
   [|79,18,3,6,8,6,2,9,17,24|...|]; % expected gain per zone
4 set of int: Types = 1..t; % type i covers i by i zones
5 array[Types] of int: Cost; % maintenance costs
6 set of int: Ant = 1..antennae;
7 % Variables (for upper-left coordinates) and constraints:
8 array[Ant] of var Zones: X; % X[a] = x-coordinate of a
9 array[Ant] of var Zones: Y; % Y[a] = y-coordinate of a
10 array[Ant] of var Types: Type; % Type[a] = type of a
11 constraint diffn(X,Y,Type,Type); % no coverage overlaps
12 constraint forall(a in Ant)
   (X[a]+Type[a] <= z+1 \ / \ Y[a]+Type[a] <= z+1);
13 % Objective:
14 array[Ant] of var 0..sum(ExpGain): Gain; % Gain[a]=gain of a
15 constraint forall(a in Ant)(Gain[a] = sum(x,y in Zones)
   (ExpGain[x,y] * (X[a] <= x \ / \ x < X[a]+Type[a] \ /
   Y[a] <= y \ / \ y < Y[a]+Type[a])));
16 var 0..(antennae*max(Cost)): cost; % total maintenance cost
17 constraint cost = sum(a in Ant)(Cost[Type[a]])
18 solve maximize sum(Gain) - cost;
Model with Pre-Computation

\[ \text{array}[\text{Zones}, \text{Zones}, \text{Types}] \text{ of int: NetGain =} \]
\[ \text{array3d}(\text{Zones}, \text{Zones}, \text{Types}, [\text{sum}(w, h \text{ in } 0..\text{Type}[t]-1 \]
\[ \text{where } x+w \leq z \backslash\ y+h \leq z)(\text{ExpGain}[x+w, y+h]) - \text{Cost}[t] \]
\[ | x, y \text{ in Zones, } t \text{ in Types}); \]
\[ \text{solve maximize } \sum(a \text{ in Ant}) (\text{NetGain}[X[a], Y[a], Type[a]]); \]

This model yields better inference and faster solving.

Solving to optimality with Gecode (CP) for \( z=10 \):

<table>
<thead>
<tr>
<th>pre-computation</th>
<th>antennae</th>
<th>t</th>
<th># nodes</th>
<th>seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>without</td>
<td>1</td>
<td>4</td>
<td>927</td>
<td>0.007</td>
</tr>
<tr>
<td>with</td>
<td>1</td>
<td>4</td>
<td>65</td>
<td>0.001</td>
</tr>
<tr>
<td>without</td>
<td>2</td>
<td>4</td>
<td>11,445,833</td>
<td>106.936</td>
</tr>
<tr>
<td>with</td>
<td>2</td>
<td>4</td>
<td>361</td>
<td>0.005</td>
</tr>
<tr>
<td>without</td>
<td>3</td>
<td>4</td>
<td>timeout</td>
<td>timeout</td>
</tr>
<tr>
<td>with</td>
<td>3</td>
<td>4</td>
<td>961</td>
<td>0.015</td>
</tr>
<tr>
<td>with</td>
<td>5</td>
<td>4</td>
<td>188,844</td>
<td>2.642</td>
</tr>
</tbody>
</table>
Outline

1. Black-Hole Patience
2. Antenna Placement
3. Warehouse Location
4. Sport Scheduling
The Warehouse Location Problem (WLP)

A company considers opening warehouses at some candidate locations in order to supply its existing shops:

- Each candidate warehouse has the same maintenance cost.
- Each candidate warehouse has a supply capacity, which is the maximum number of shops it can supply.
- The supply cost to a shop depends on the warehouse.

Determine which candidate warehouses actually to open, and which of them supplies which shops, so that:

1. Each shop is supplied by exactly one actually opened warehouse.
2. Each actually opened warehouse supplies a number of shops at most equal to its capacity.
3. The sum of the actually incurred maintenance costs and supply costs is minimal.
WLP: Sample Instance Data

Shops = \{Shop_1, Shop_2, \ldots, Shop_{10}\}


maintCost = 30

<table>
<thead>
<tr>
<th>Capacity</th>
<th>Berlin</th>
<th>London</th>
<th>Ankara</th>
<th>Paris</th>
<th>Rome</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SupplyCost</th>
<th>Berlin</th>
<th>London</th>
<th>Ankara</th>
<th>Paris</th>
<th>Rome</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shop_1</td>
<td>20</td>
<td>24</td>
<td>11</td>
<td>25</td>
<td>30</td>
</tr>
<tr>
<td>Shop_2</td>
<td>28</td>
<td>27</td>
<td>82</td>
<td>83</td>
<td>74</td>
</tr>
<tr>
<td>Shop_3</td>
<td>74</td>
<td>97</td>
<td>71</td>
<td>96</td>
<td>70</td>
</tr>
<tr>
<td>Shop_4</td>
<td>2</td>
<td>55</td>
<td>73</td>
<td>69</td>
<td>61</td>
</tr>
<tr>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
</tr>
<tr>
<td>Shop_{10}</td>
<td>47</td>
<td>65</td>
<td>55</td>
<td>71</td>
<td>95</td>
</tr>
</tbody>
</table>
WLP Model 1: Decision Variables

Automatic enforcement of the total-function constraint (1):

\[
\text{Supplier} = \begin{bmatrix}
\text{Shop}_1 & \text{Shop}_2 & \cdots & \text{Shop}_{10}
\end{bmatrix}
\begin{bmatrix}
\in \text{Whs} & \in \text{Whs} & \cdots & \in \text{Whs}
\end{bmatrix}
\]

\text{Supplier}[s] \text{ denotes the supplier warehouse for shop } s.\]

Variables redundant with \text{Supplier}, but not mutually:

\[
\text{Open} = \begin{bmatrix}
\text{Berlin} & \text{London} & \text{Ankara} & \text{Paris} & \text{Rome}
\end{bmatrix}
\begin{bmatrix}
\in 0..1 & \in 0..1 & \in 0..1 & \in 0..1 & \in 0..1
\end{bmatrix}
\]

\text{Open}[w] = 1 \text{ if and only if (iff) warehouse } w \text{ is opened.} \]

\(\Rightarrow\) \text{ Our chosen array names always reflect total functions.} \]
solve minimize maintCost * sum(Open)
+sum(s in Shops)(SupplyCost[s, Supplier[s]])

The first term is the total maintenance cost, expressed as the product of the warehouse maintenance cost by the number of actually opened warehouses.

The second term is the total supply cost, expressed as the sum over all shops of their actually incurred supply costs.

Notice the implicit use of the element predicate, as the index Supplier[s] is a decision variable.

If warehouse \( w \) has maintenance cost \( \text{MaintCost}[w] \), then the first term becomes

\[
\text{sum}(w \text{ in Whs})(\text{MaintCost}[w] * \text{Open}[w]).
\]
WLP Model 1: Channelling Constraint

One-way channelling constraint from the Supplier variables to its redundant Open variables:

\[ \text{forall}(s \text{ in } \text{Shops})(\text{Open}[\text{Supplier}[s]] = 1) \]

The supplier warehouse of each shop is actually opened.

Notice the implicit use of the element predicate, as the index Supplier[s] is a decision variable.

How do the remaining Open[w] variables become 0?
Upon minimisation.
WLP Model 1: Channelling Constraint

**Alternative:** Two-way channelling constraint between the `Supplier` variables and its redundant `Open` variables:

\[
\text{forall}(w \text{ in Whs}) \quad (\text{Open}[w] = (\text{exists}(s \text{ in Shops})(\text{Supplier}[s]=w)))
\]

A warehouse is opened iff there exists a shop it supplies.

Make experiments to find out which channelling is better. We will revisit this issue in Topic 8: Inference & Search in CP & LCG, and in Topic 9: Modelling for CBLS.

Nothing changes if `Open` is an array of Boolean variables.
WLP Model 1: Capacity Constraint

Capacity constraint (2):

\[
global\_cardinality\_low\_up\_closed
(Supplier, Whs, [0 \mid w \text{ in } Whs], Capacity)
\]

Each actually opened warehouse is a supplier of a number of shops at most equal to its capacity.

Which symmetries are there?

- There are no problem symmetries.
- We introduced no symmetries into the model.
- There may be instance symmetries: indistinguishable shops, or indistinguishable warehouses, or both.
WLP Model 2

Drop the array `Open` of redundant decision variables as well as its channelling constraint, and reformulate the first term of the objective function as follows:

\[
\text{maintCost} \ast \\
\sum(w \text{ in } \text{Whs})(\exists(s \text{ in } \text{Shops})(\text{Supplier}[s] = w))
\]

We can alternatively use the `nvalue` constrained function:

\[
\text{maintCost} \ast \\
n\text{value}(\text{Supplier})
\]

This alternative formulation cannot be generalised for warehouse-specific maintenance costs. For a speed comparison, see Topic 8: Inference & Search in CP & LCG.

Redundancy elimination may pay off, but it may just as well be the converse.

But this is hard to guess, as human intuition may be weak.
WLP Model 3: Decision Variables

No automatic enforcement of total-function constraint (1):

\[
\text{Supply} = \begin{bmatrix}
\text{Shop}_1 & \text{Shop}_2 & \cdots & \text{Shop}_{10} \\
\vdots & \vdots & \ddots & \vdots \\
\text{Supply}[s, w] &= 1 \text{ iff shop } s \text{ is supplied by warehouse } w.
\end{bmatrix}
\]

Redundant decision variables (as in Model 1):

\[
\text{Open} = \begin{bmatrix}
\text{Berlin} & \text{London} & \text{Ankara} & \text{Paris} & \text{Rome} \\
\in 0..1 & \in 0..1 & \in 0..1 & \in 0..1 & \in 0..1
\end{bmatrix}
\]

\[
\text{Open}[w] = 1 \text{ if and only if warehouse } w \text{ is opened.}
\]
WLP Model 3: Objective

The objective can now be expressed in linear fashion:

\[
\text{solve minimize } \\
maintCost \times \text{sum(Open)} \\
+ \\
\text{sum(s in Shops, w in Whs)} \\
(\text{Supply}[s,w] \times \text{SupplyCost}[s,w])
\]

The first term is the total maintenance cost, expressed (as in Model 1) as the product of the warehouse maintenance cost by the number of actually opened warehouses.

The second term is the total supply cost, expressed as the sum over all shops and warehouses of their actually incurred supply costs: each parameter $\text{SupplyCost}[s,w]$ is weighted by the decision variable $\text{Supply}[s,w]$. 
WLP Model 3: Constraints

The total-function constraint (1) now needs to be modelled, and can be expressed in linear fashion without count:

\[ \forall (s \in \text{Shops}) (\sum (\text{Supply}[s,\ldots]) = 1) \]

Each shop is supplied by exactly one actually opened warehouse.
WLP Model 3: Constraints (end)

Capacity constraint (2), in isolation:

\[
\text{forall} (w \text{ in } \text{Whs}) \\
\quad (\text{sum}(\text{Supply}[..,w]) \leq \text{Capacity}[w])
\]

Two-way channelling constraint, in isolation:

\[
\text{forall} (w \text{ in } \text{Whs}) \\
\quad (\text{sum}(\text{Supply}[..,w]) > 0 \leftrightarrow \text{Open}[w] = 1)
\]

or, one-way without reification, upon exploiting minimisation:

\[
\text{forall} (w \text{ in } \text{Whs}) \\
\quad (\text{forall}(s \text{ in } \text{Shops})(\text{Supply}[s,w] \leq \text{Open}[w]))
\]

Capacity (2) & one-way channelling constraints combined:

\[
\text{forall} (w \text{ in } \text{Whs}) \\
\quad (\text{sum}(\text{Supply}[..,w]) \leq \text{Capacity}[w] \times \text{Open}[w])
\]

All constraints are linear (in)equalities: this is an IP model!
Outline

1. Black-Hole Patience
2. Antenna Placement
3. Warehouse Location
4. Sport Scheduling
The Sport Scheduling Problem (SSP)

Find schedule in $\text{Periods} \times \text{Weeks} \rightarrow \text{Teams} \times \text{Teams}$ for

- $|\text{Teams}| = n$ and $n$ is even
- $|\text{Weeks}| = n-1$
- $|\text{Periods}| = n/2$ periods per week

subject to the following constraints:

1. Each possible game is played exactly once.
2. Each team plays exactly once per week.
3. Each team plays at most twice per period.

Idea for a model, and a solution for $n=8$

<table>
<thead>
<tr>
<th></th>
<th>Wk 1</th>
<th>Wk 2</th>
<th>Wk 3</th>
<th>Wk 4</th>
<th>Wk 5</th>
<th>Wk 6</th>
<th>Wk 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>P 1</td>
<td>1 vs 2</td>
<td>1 vs 3</td>
<td>2 vs 6</td>
<td>3 vs 5</td>
<td>4 vs 7</td>
<td>4 vs 8</td>
<td>5 vs 8</td>
</tr>
<tr>
<td>P 2</td>
<td>3 vs 4</td>
<td>2 vs 8</td>
<td>1 vs 7</td>
<td>6 vs 7</td>
<td>6 vs 8</td>
<td>2 vs 5</td>
<td>1 vs 4</td>
</tr>
<tr>
<td>P 3</td>
<td>5 vs 6</td>
<td>4 vs 6</td>
<td>3 vs 8</td>
<td>1 vs 8</td>
<td>1 vs 5</td>
<td>3 vs 7</td>
<td>2 vs 7</td>
</tr>
<tr>
<td>P 4</td>
<td>7 vs 8</td>
<td>5 vs 7</td>
<td>4 vs 5</td>
<td>2 vs 4</td>
<td>2 vs 3</td>
<td>1 vs 6</td>
<td>3 vs 6</td>
</tr>
</tbody>
</table>
The Sport Scheduling Problem (SSP)

Find schedule in $\text{Periods} \times \text{Weeks} \rightarrow \text{Teams} \times \text{Teams}$ for

- $|\text{Teams}| = n$ and $n$ is even
- $|\text{Weeks}| = n-1$
- $|\text{Periods}| = n/2$ periods per week

subject to the following constraints:

1. Each possible game is played exactly once.
2. Each team plays exactly once per week.
3. Each team plays at most twice per period.

Idea for a model, and a solution for $n=8$, with a dummy week $n$ of duplicate games:

<table>
<thead>
<tr>
<th></th>
<th>Wk 1</th>
<th>Wk 2</th>
<th>Wk 3</th>
<th>Wk 4</th>
<th>Wk 5</th>
<th>Wk 6</th>
<th>Wk 7</th>
<th>Wk 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>P 1</td>
<td>1 vs 2</td>
<td>1 vs 3</td>
<td>2 vs 6</td>
<td>3 vs 5</td>
<td>4 vs 7</td>
<td>4 vs 8</td>
<td>5 vs 8</td>
<td>6 vs 7</td>
</tr>
<tr>
<td>P 2</td>
<td>3 vs 4</td>
<td>2 vs 8</td>
<td>1 vs 7</td>
<td>6 vs 7</td>
<td>6 vs 8</td>
<td>2 vs 5</td>
<td>1 vs 4</td>
<td>3 vs 5</td>
</tr>
<tr>
<td>P 3</td>
<td>5 vs 6</td>
<td>4 vs 6</td>
<td>3 vs 8</td>
<td>1 vs 8</td>
<td>1 vs 5</td>
<td>3 vs 7</td>
<td>2 vs 7</td>
<td>2 vs 4</td>
</tr>
<tr>
<td>P 4</td>
<td>7 vs 8</td>
<td>5 vs 7</td>
<td>4 vs 5</td>
<td>2 vs 4</td>
<td>2 vs 3</td>
<td>1 vs 6</td>
<td>3 vs 6</td>
<td>1 vs 8</td>
</tr>
</tbody>
</table>
SSP Model 1: Data

Parameter:

- int: n; assert(n>1 \&\& n \mod 2 =0, "Odd n")

Useful Ranges, enumeration, and set:

- Teams = 1..n
- Weeks = 1..(n-1)
- ExtendedWeeks = 1..n
- Periods = 1..(n \div 2)
- Slots = {one,two}
- Games = {f*n+s | f,s in Teams where f<s}, thereby breaking some symmetries, such that the game between teams f and s is uniquely identified by the natural number f * n + s.

Example: For n=4, we get Games={6,7,8,11,12,16}. 
SSP Model 1: Decision Variables

A 3d matrix $\text{Team}[\text{Periods, ExtendedWeeks, Slots}]$ of variables in Teams, denoted $T$ below, over a schedule extended by a dummy week where teams play fictitious duplicate games in the period where they would otherwise play only once, thereby transforming constraint (3) into:

$$(3') \text{ Each team plays exactly twice per period.}$$

Predicate $\text{global_cardinality_low_up_closed}$ need not be used and can be replaced by a stronger predicate.

Team =

$$
\begin{array}{cccccccc}
\text{Wk 1} & \text{Wk 2} & \cdots & \cdots & \text{Wk } n-1 & \text{Wk } n \\
\text{one} & \text{two} & \cdots & \cdots & \text{one} & \text{two} \\
\text{P 1} & \in T & \in T & \cdots & \cdots & \in T & \in T \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
\text{P n/2} & \in T & \in T & \cdots & \cdots & \in T & \in T \\
\end{array}
$$

$\text{Team}[p, w, s]$ is the numeric name of the team that plays in period $p$ of week $w$ in game slot $s$. 
SSP Model 1: Constraints

Twice-per-period constraint (3’):

\[
\text{forall}(p \ \text{in} \ \text{Periods}) \\
(\text{global_cardinality_closed} \\
(Team[p,\ldots,\ldots], \ \text{Teams}, \ [2 \ | \ i \ \text{in} \ 1..n]))
\]

In each period, each team name occurs exactly twice within the slots of the weeks in Team.

Once-per-week constraint (2):

\[
\text{forall}(w \ \text{in} \ \text{ExtendedWeeks}) \\
(\text{alldifferent}(Team[\ldots,w,\ldots]))
\]

In each week, incl. the dummy week, there are no duplicate team names within the slots of the periods in Team.
Try to state the each-game-once constraint (1) using Team!

Declare a 2d matrix $\text{Game}[\text{Periods}, \text{Weeks}]$ of decision variables in $\text{Games}$ over the non-extended weeks:

$$
\text{Game} =
\begin{array}{c|c|c}
\text{Period 1} & \text{Week 1} & \cdots & \text{Week } n - 1 \\
\vdots & \in \text{Games} & \cdots & \in \text{Games} \\
\text{Period } n/2 & \vdots & \ddots & \vdots \\
\end{array}
\begin{array}{c|c|c}
\in \text{Games} & \cdots & \in \text{Games} \\
\end{array}
$$

$\text{Game}[p, w]$ is the game played in period $p$ of week $w$.

The 2d $\text{Game}$ is mutually redundant with the first $n - 1$ 2d columns of the 3d $\text{Team}$, which is over the extended weeks.
Each-game-once constraint (1):

\texttt{alldifferent}(\text{Game})

There are no duplicate game numbers in \textit{Game}.

Two-way channelling constraint (but rather use \textit{table}:

\texttt{forall}(p \text{ in Periods}, w \text{ in Weeks})

\begin{align*}
& (\text{Team}[p, w, \text{one}] \times n + \text{Team}[p, w, \text{two}] = \text{Game}[p, w])
\end{align*}

The game number in \textit{Game} of each period and week corresponds to the teams scheduled at that time in \textit{Team}.

Constraints (2) and (3') are hard to formulate using \textit{Game}.

Add the symmetry-breaking constraints of slide 29 of Topic 5: Symmetry.
A round-robin schedule suffices to break many of the remaining symmetries:

- Restrict the games of the first week to the set 
  \[ \{1 \text{ vs } 2\} \cup \{t + 1 \text{ vs } n + 2 - t \mid 1 < t \leq n/2\} \]
- For the remaining weeks, transform each game \(f\) vs \(s\) of the previous week into a game \(f'\) vs \(s'\), where

\[
f' = \begin{cases} 
1 & \text{if } f = 1 \\
2 & \text{if } f = n \\
f + 1 & \text{otherwise}
\end{cases}
\]

\[
s' = \begin{cases} 
2 & \text{if } s = n \\
s + 1 & \text{otherwise}
\end{cases}
\]

The constraints (1) and (2) are now automatically enforced: we must determine the period of each game, not its week!
Interested in More Details?

For more details on WLP & SSP and their modelling, see:

