



## Some Results About Airline Scheduling

### Sliding Calculations

#### Introduction

Scheduling constraints occasionally require the presence of some “feature” over all intervals of fixed width, such as the following TWA<sup>1</sup> rule for rostering airline flight attendants:

*Rosters of one month's duration must have at least 24 consecutive hours free of duty among any 7 consecutive days*

Dividing the month into 720 hours, a roster may be represented by a sequence of 720 binary digits (0s and 1s), where 0 represents rest and 1 represents work.

Then this rule can be stated in the following way:

*Sequences of 720 binary digits must have at least 24 consecutive 0s among any consecutive 168 binary digits*

A simple recipe for verifying this rule is the following: Let  $N = 720$ ,  $M = 168$  and  $L = 24$  and label as *basic* any consecutive sequence of  $L$  or more 0s not contained in any larger such sequence.<sup>2</sup>

So our rule is violated if there are no basic sequences, or the first one begins on or later than the  $(M - L + 2)$ -th digit, or the last one ends on or before the  $(N - M + L - 1)$ -th digit, or two consecutive ones are at least  $M - 2L + 3$  digits apart. Otherwise, the rule holds.

This note shows that the above recipe is true in general, and provides an algorithm for implementing it. Later some additional questions are posed.

#### Definitions

For any sequence  $S = \langle s_i, i = 1, \dots, N \rangle$ , its *length*  $\text{len}(S)$  is  $N$ .

If  $S = \langle s_i, i = M, \dots, N \rangle$  is any sequence, its *starting position* is  $M$ .

---

<sup>1</sup> Named after the airline that used this rule

<sup>2</sup> ie. bounded by a pair of 1s

If  $S = \langle s_i, i = M, \dots, N \rangle$  and  $T = \langle t_i, i = M', \dots, N' \rangle$  are both sequences, then  $S$  is a *subsequence* of  $T$ , denoted  $S \subseteq T$ , if the following holds:

$$M' \leq M \leq N \leq N'$$

$$s_i = t_i \text{ for } i = M, \dots, N$$

In other words, subsequences are obtained from sequences by dropping some their beginning and ending elements. Clearly  $\subseteq$  is a partial order.

If  $S \subseteq T$ , then  $S$  is said to be *contained* in  $T$ . Subsequences  $S$  of a fixed sequence  $T$  will sometimes be identified by their terminal indices, eg.  $[M, N] = \langle t_i, i = M, \dots, N \rangle$ .

A *work period*  $W$  is any sequence  $\langle w_i, i = 1, \dots, N \rangle$  of binary digits  $w_i, i = 1, \dots, N$ .

Any subsequence of a work period containing just 0s is called a *rest period*. A rest period is *maximal* if it is not contained in a larger rest period.

In the following work period of length 22, some of the rest periods are shown, one per line. Those that are maximal are shaded.

0	0	0	1	0	0	1	1	1	0	1	0	0	1	1	0	0	1	0	1	0	0
0																					
0	0																				
0	0	0																			
	0	0																			
				0																	
					0																
				0	0																
									0												
										0	0										

An algorithm is required for the following procedure:

### Sliding(W,N,M,L)

Return T if work period  $W$  of length  $N$  has every subsequence  $S$  of length  $M$  containing at least  $L$  consecutive 0s. Otherwise, return F.

This procedure assumes that  $1 \leq L \leq M \leq N$ .

For example, the procedure  $\text{Sliding}(W, 720, 168, 24)$  will verify the TWA rule.

For a given  $W, N, M, L$  in  $\text{Sliding}(W, N, M, L)$ , a maximal rest period is called *basic* if its length is at least  $L$ . Clearly any pair of distinct basic periods are separated by at least one 1 (otherwise

they wouldn't be maximal). Furthermore, basic periods are linearly ordered by their starting positions.

In the above example, if  $M = 2$ , the basic periods are those shown below:

0	0	0	1	0	0	1	1	1	0	1	0	0	1	1	0	0	1	0	1	0	0
0	0	0																			
				0	0																
										0	0										
														0	0						
																				0	0

On the other hand, if  $M = 3$ , only the first maximal rest period is basic. If  $M = 1$ , then they are all basic.

The following theorem is useful for programming the Sliding procedure.

**Theorem 1**

Sliding( $W,N,M,L$ ) fails if and only (iff) if at least one of the following holds:

- (1) There are no basic periods
- (2) The starting position  $s$  of the first basic period satisfies:

$$s \geq M - L + 2$$

- (3) The ending position  $e$  of the last basic period satisfies:

$$e \leq N - M + L - 1$$

- (4) The ending position  $e$  of a basic period and the starting position  $s$  of the next basic period satisfy:

$$s - e \geq M - 2L + 3$$

In other words, the procedure will fail iff a sufficiently large gap exists between a pair of adjacent basic periods, or the first one starts too late, or the last one ends too soon, or there are no basic periods.

**Proof**

We will show the left hand side implies the right, and vice versa.

LHS  $\rightarrow$  RHS

Suppose Sliding( $W,N,M,L$ ) fails.

Then there is a subsequence  $S = [s_0, s_1]$  of length  $M$  which contains at most  $L - 1$  consecutive 0s.

Assume there is at least one basic period. It remains to show that (2) or (3) or (4) holds.

Let  $\mathcal{P}$  be the set of basic periods  $P = [p_0, p_1]$  such that  $p_0 < s_0$  and let  $\mathcal{Q}$  be the set of basic periods  $Q = [q_0, q_1]$  such that  $s_0 \leq q_0$ . Obviously each basic period belongs to exactly one of  $\mathcal{P}$  or  $\mathcal{Q}$ . Since there is at least one basic period then exactly one of the following holds:

- (a)  $\mathcal{P}$  is non-empty but  $\mathcal{Q}$  is empty
- (b)  $\mathcal{P}$  is empty but  $\mathcal{Q}$  is non-empty
- (c)  $\mathcal{P}$  is non-empty and  $\mathcal{Q}$  is non-empty

Suppose (a) holds. Choose  $P = [p_0, p_1] \in \mathcal{P}$  with the largest starting point  $p_0$ . Since at most  $L - 1$  elements of  $P$  can belong to  $S$ ,  $p_1 - s_0 + 1 \leq L - 1$ . Hence  $p_1 \leq s_0 - 1 + L - 1 \leq (N - M + 1) - 1 + L - 1 = N - M + L - 1$  so (3) holds.

Suppose (b) holds. Choose  $Q = [q_0, q_1] \in \mathcal{Q}$  with the smallest starting point  $q_0$ . Since at most  $L - 1$  elements of  $Q$  can belong to  $S$ ,  $s_1 - q_0 + 1 \leq L - 1$ . Hence  $q_0 \geq s_1 + 1 - L + 1 = s_0 + M - 1 + 1 - L + 1 \geq 1 + M - 1 + 1 - L + 1 = M - L + 2$  so (2) holds.

Suppose (c) holds. Choose  $P = [p_0, p_1] \in \mathcal{P}$  with the largest starting point  $p_0$  and  $Q = [q_0, q_1] \in \mathcal{Q}$  with the smallest starting point  $q_0$ . Then  $P$  is the immediate predecessor of  $Q$  (see Fig. 1). Since the length of  $S$  is  $M$ ,  $s_0 + M - 1 = s_1$ , so by rearranging and adding terms:  $q_0 - p_1 = M - (p_1 - s_0) - (s_1 - q_0) + 1$ . But  $p_1 - s_0 + 1 \leq L - 1$  if  $p_1 \geq s_0$  since there can be at most  $L - 1$  elements between  $s_0$  and  $p_1$ . If  $p_1 < s_0$  the inequality holds trivially. For similar reasons,  $s_1 - q_0 + 1 \leq L - 1$ . From this  $q_0 - p_1 \geq M - (L - 2) - (L - 2) + 1$  so (4) follows.

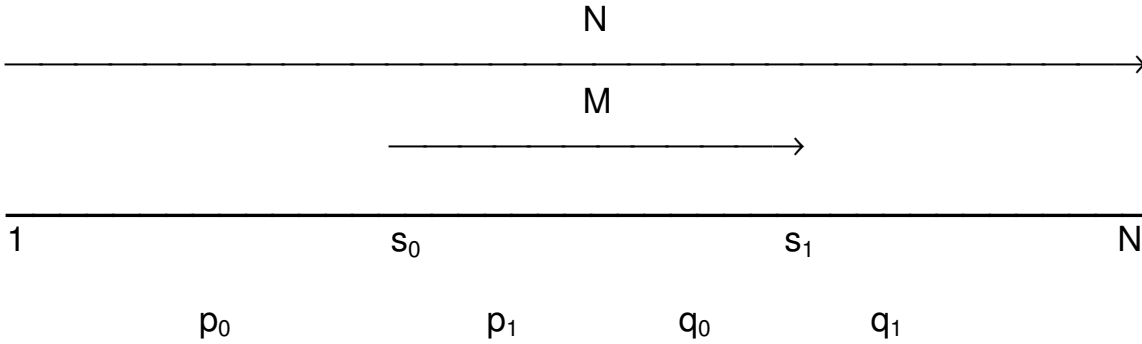


Fig. 1

RHS  $\rightarrow$  LHS

If (1) holds then Sliding( $W, N, M, L$ ) fails trivially.

Suppose (2) holds for the first basic period  $Q = [q_0, q_1]$ , ie.  $q_0 \geq M - L + 2$ . In particular,  $M - q_0 + 1 \leq L - 1$ . Consider  $S = [1, M]$ , which is a subsequence of length  $M$ . Let  $n$  be the number of members of  $Q$  contained in  $S$ . Then  $S$  will contain  $L$  consecutive 0s iff  $n \geq L$  since  $Q$  is the first basic period. But  $n \leq \max\{0, M - q_0 + 1\} \leq \max\{0, L - 1\}$  so  $S$  cannot contain  $L$  consecutive 0s. Hence Sliding( $W, N, M, L$ ) fails.

Suppose (3) holds for the last basic period  $P = [p_0, p_1]$ , ie.  $p_1 \leq N - M + L - 1$ . In particular,  $p_1 - N + M \leq L - 1$ . Consider  $S = [N - M + 1, N]$ , which is a subsequence of length  $M$ . Let  $n$  be the number of members of  $P$  contained in  $S$ . Then  $P$  will contain  $L$  consecutive 0s iff  $n \geq L$  since  $P$  is the last basic period. But  $n \leq \max\{0, p_1 - (N - M + 1) + 1\} \leq \max\{0, L - 1\}$  so  $S$  cannot contain  $L$  consecutive 0s. Hence Sliding( $W, N, M, L$ ) fails.

Suppose (4) holds for a basic period  $P = [p_0, p_1]$  and the next basic period  $Q = [q_0, q_1]$ , ie.  $q_0 - p_1 \geq M - 2L + 3$ . Consider  $S = [p_1 - L + 1, q_0 + L - 1]$ , which consists of the  $q_0 - p_1 - 1$  elements strictly between  $p_1$  and  $q_0$  along with the last  $L$  elements of  $P$  and the first  $L$  elements of  $Q$ . The length of  $S$  is  $q_0 + L - 1 - (p_1 - L + 1) + 1 = q_0 - p_1 + 2L - 1 \geq M - 2L + 3 + 2L - 1 = M + 2$ . By deleting the first and last members,  $S$  is reduced to a subsequence of length  $M$  containing  $L - 1$  consecutive 0s, but no more. Hence Sliding( $W, N, M, L$ ) fails.

### Algorithm

Input  $W, N, M, L$ .

Compute the vector  $R$  of length  $N + 1$  recursively as follows:

$$R_{N+1} = 0$$

$$R_i = (1 - W_i) * (1 + R_{i+1}) \quad i = N, N - 1, \dots, 1$$

Clearly  $R_i$  computes the longest sequence of all 0s in  $W$ , starting at the  $i$ -th position. By definition,  $R_i = 0$  if  $W_i = 1$ .

It is easy to see that the  $i$ -th position is the start of a basic period iff  $R_i \geq L$  and  $W_{i-1} = 1$  ( $i=1, \dots, N$ )<sup>3</sup>. So when  $R$  is being computed, one may quickly calculate if the start of the next basic period has been reached. Always save the last one reached as the computation proceeds.

Now note that if  $i < j$  are the starting positions of adjacent basic periods, then inequality (4) in Theorem 1

$$s - e \geq M - 2L + 3$$

is equivalent to

$$j - (i + R_i - 1) \geq M - 2L + 3$$

Inequalities (2) and (3) are performed similarly. So while  $R$  is being computed one element at a time, test this inequality whenever the next basic period has been reached<sup>4</sup>. Stop and return  $F$  if any inequality is violated, otherwise return  $T$ .

## Examples

Basic periods are shaded.

$R =$ 

3	2	1	0	2	1	0	0	0	1	0	2	1	0	0	2	1	0	1	0	2	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

 0  
 $W =$ 

0	0	0	1	0	0	1	1	1	0	1	0	0	1	1	0	0	1	0	1	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

 1  
 Sliding( $W, 22, 5, 1$ ) holds

$R =$ 

3	2	1	0	2	1	0	0	0	1	0	2	1	0	0	2	1	0	1	0	2	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

 0  
 $W =$ 

0	0	0	1	0	0	1	1	1	0	1	0	0	1	1	0	0	1	0	1	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

 1  
 Sliding( $W, 22, 5, 2$ ) fails

$R =$ 

3	2	1	0	2	1	0	0	0	1	0	2	1	0	0	2	1	0	1	0	2	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

 0  
 $W =$ 

0	0	0	1	0	0	1	1	1	0	1	0	0	1	1	0	0	1	0	1	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

 1  
 Sliding( $W, 22, 5, 3$ ) fails

<sup>3</sup> Set  $W_0 = 1$

<sup>4</sup> On the first and last basic period, inequality (2) or (3) must also be tested.

## Note

Once it is known that Sliding( $W, N, M, L$ ) holds, then changing any contiguous sequence of 0s and 1s in  $W$  requires only that the above computation be performed in the “neighborhood” of those changes.

## Another Problem

Another problem is the following: If Sliding( $W, N, M, L$ ) fails, then what is the largest value  $L'$  for which it holds? Clearly the value  $L - L'$  would indicate how badly the rule is broken. In particular, it could be used to decide if the rule violation may be ignored.

This question is answered by the following algorithm which provides for any  $W$ ,  $N$ , and  $M$  the largest  $L$  such that Sliding( $W, N, M, L$ ) holds.

In what follows, a given binary sequence  $W$  of length  $N$  is given, along with a fixed length  $M$ . In particular,  $W = W_1, W_2, \dots, W_N$ , where  $W_i = 0, 1$  and  $1 \leq M \leq N$ .

From now on we will assume that  $M \geq 2$ , since otherwise the problem is trivial. This will eliminate special cases in the inequalities presented below.

## Definitions

A *period* is any maximal subsequence of 0s (*rest*) or maximal subsequence of 1s (*work*) in  $W$ .

We may easily represent  $W$  by its corresponding sequence  $X$  of period lengths  $X_i$  (where  $X_i$  is the length of the  $i$ -th period):

$$X = X_1, X_2, \dots, X_p$$

In order to flag what type of period each  $X_i$  represents, the following function  $T$  is used:

$$T(i) \equiv T_0 + i - 1 \pmod{2} \text{ for } i = 1, \dots, P$$

Here  $T(i) = 0$  or  $1$ , depending on whether  $X_i$  is a rest or work period ( $P$  denotes the number of periods). Note that the period types alternate between 0 and 1, and that the sum of the  $X_i$  is  $N$ .

Any subsequence  $s$  of  $W$  whose length is  $M$  is called a *slider*.

Intuitively, we want to move a slider from left to right, always computing the greatest rest period contained within it, while retaining the lowest value  $L$  found among all positions. This will answer our question.

The trick is to do it quickly, using  $X$  instead of  $W$ . For example, if a slider starts at the beginning of the  $i$ -th period and ends at the end of the  $k$ -th, the greatest rest period for that slider is simply the largest of those rest periods between the two given periods (ie, the largest  $X_j$  whose  $T(j) = 0$ , for  $j$  between  $i$  and  $k$ ).

All of what happens below concerns sliders which don't start and stop on period boundaries. Fortunately, we can still compute the largest rest period contained within them by continuing to use just  $X$  and  $M$ .

A subsequence  $\sigma = X_i, X_{i+1}, \dots, X_k$  of  $X$  is called *basic* if the following *basic inequalities* hold:

If  $i = k$  or  $i + 1 = k$ :

$$M \leq \sum_{i \leq j \leq k} X_j$$

If  $i + 1 < k$ :

$$M \leq \sum_{i \leq j \leq k} X_j$$

$$\sum_{i < j < k} X_j \leq M - 2$$

The periods  $X_j$  ( $i < j < k$ ) are called the *inner* periods of  $\sigma$ , while  $X_i, X_k$  are its *outer* periods.

The first inequality in either case guarantees that a slider can be found which starts on or after the start of the  $i$ -th period, and ends on or before the end of the  $k$ -th period. Adding the last inequality in either case guarantees that such a slider can be constrained to begin within the  $i$ -th period, and end within the  $k$ -th period (since the "inner" periods must "miss" at least two elements of  $W$ ). Any such slider is said to *span*  $\sigma$ .

Clearly the following properties hold:

- Every slider spans exactly one basic subsequence of periods  $\sigma$
- Every basic subsequence of periods  $\sigma$  is spanned by at least one slider

Let  $\mathcal{S}_\sigma$  represent the set of all sliders spanning  $\sigma$ . Then by the above properties:

$$\mathcal{S} = \cup_{\sigma} \mathcal{S}_\sigma$$

represents a disjoint union of non-empty sets of sliders containing all possible sliders. That is, each slider is contained in exactly one of the  $\mathcal{S}_\sigma$ .

We will show that the  $\sigma$  can be enumerated quickly, and that the required number  $L$  may be determined by doing a simple computation on each  $\mathcal{S}_\sigma$  during that enumeration. In other words, by doing one quick pass through the basic subsequence of periods, the problem may be solved.

For each slider  $s$ , let  $|s|$  be the longest rest contained within it (ie. the length of the largest subsequence of 0s of  $s$ ).

For each basic subsequence of periods  $\sigma$ , let  $|\delta_\sigma|$  be the smallest  $|s|$  of all  $s$  spanning it.

Clearly the following identity holds:

The largest  $L$  such that Sliding( $W, N, M, L$ ) holds is the smallest  $|\delta_\sigma|$  for any basic subsequence of periods  $\sigma$ .

So to compute  $L$  from the above identity, we need only enumerate each  $\sigma$ , compute  $|\delta_\sigma|$  and keep the smallest value.

The following theorem shows how to compute  $|\delta_\sigma|$ .

### **Theorem**

Suppose  $\sigma = X_i, X_{i+1}, \dots, X_k$  is a basic subsequence of periods.

Let  $L_0$  be the largest  $X_j$  with  $T(j) = 0$  for  $i < j < k$ , and let  $M_0 = \sum_{i < j < k} X_j$ .<sup>5</sup>

Let $L_1 = M$	if $i = k$ and $T(i) = 0$
$= 0$	if $i = k$ and $T(i) = 1$
$= M - M_0 - \min(X_i, X_k, \lfloor (M - M_0) / 2 \rfloor)$	if $i < k$ and $T(i) = 0$ and $T(k) = 0$ <sup>6</sup>
$= M - M_0 - \min(M - M_0 - 1, X_k)$	if $i < k$ and $T(i) = 0$ and $T(k) = 1$
$= M - M_0 - \min(M - M_0 - 1, X_i)$	if $i < k$ and $T(i) = 1$ and $T(k) = 0$
$= 0$	if $i < k$ and $T(i) = 1$ and $T(k) = 1$

Then  $|\delta_\sigma| = \max(L_0, L_1)$ .

### **Proof**

By definition,  $|\delta_\sigma|$  is the smallest  $|s|$  for all  $s$  spanning  $\sigma$ . Suppose  $s$  is any spanning slider for  $\sigma$ . Let  $L_1$  be the length of the largest subsequence of 0s of  $s$  contained in the  $i$ -th or  $k$ -th periods. Then clearly  $|s| = \max(L_0, L_1)$ .

It remains to show that  $L_1$  may be computed by the above formulae when  $|s|$  is minimal<sup>7</sup>. Note that any slider  $s$  is determined by the number  $s_i, s_k$  of 0s in the  $i$ -th and  $k$ -th periods, and they are characterized by the following inequalities:

$$\begin{aligned}
 1 &\leq s_i \leq X_i \\
 1 &\leq s_k \leq X_k \\
 s_i + s_k &= M - M_0 \\
 s_i, s_k &\text{ integers}
 \end{aligned}$$

<sup>5</sup> If there are no such  $T(j)$  then set  $L_0 = 0$ . Similarly, if  $i = k$  or  $i = k+1$  then set  $M_0 = 0$ .

<sup>6</sup> By definition,  $\lfloor X \rfloor =$  greatest integer  $\leq X$  and  $\lceil X \rceil =$  least integer  $\geq X$ .

<sup>7</sup> Note that  $L_0$  is fixed for all  $s$ .

So, strictly speaking,

$$\begin{aligned}
 L_1 &= \min_{s \text{ slider of } \sigma} \max(s_i, s_k) \\
 &= \min_{\substack{1 \leq s_i \leq X_i \\ 1 \leq s_k \leq X_k \\ s_i + s_k = M - M_0 \\ s_i, s_k \text{ integers}}} \max(s_i, s_k)
 \end{aligned}$$

In other words, to compute  $L_1$ , choose  $M - M_0$  elements from the outer periods of  $\sigma$  in order to build a spanning slider whose maximum rest in those periods is minimal. Recall that  $M - M_0 \geq 2$ .

Cases one, two and six are obvious.

In the third case, there are two possibilities:

$$(a) \min(X_i, X_k) < \lfloor (M - M_0) / 2 \rfloor$$

$$(b) \min(X_i, X_k) \geq \lfloor (M - M_0) / 2 \rfloor$$

If (a) holds, it follows that a spanning slider  $s$  may be constructed using all 0s of the smaller outer period, say  $X_i$ , while a greater number of 0s may be chosen from the larger  $X_k$ . But this greater number must be  $L_1$ , since sliding this spanner to the right will result in larger values of  $|s|$ , and all other sliders can be obtained this way. So

$$\begin{aligned}
 L_1 &= M - M_0 - X_i \\
 &= M - M_0 - \min(X_i, X_k) \\
 &= M - M_0 - \min(X_i, X_k, \lfloor (M - M_0) / 2 \rfloor)
 \end{aligned}$$

If (b) holds, then a spanning slider  $s$  may be constructed using  $\lfloor (M - M_0) / 2 \rfloor$  elements from one of the outer periods, and  $\lceil (M - M_0) / 2 \rceil$  from the other. All other “unbalanced” sliders would result in larger values of  $|s|$ , so

$$\begin{aligned}
 L_1 &= \lceil (M - M_0) / 2 \rceil \\
 &= M - M_0 - \lfloor (M - M_0) / 2 \rfloor \\
 &= M - M_0 - \min(X_i, X_k, \lfloor (M - M_0) / 2 \rfloor)
 \end{aligned}$$

The fourth and fifth cases are similar.

### Algorithm

Given  $W$ ,  $N$  and  $M$ .

If  $M = 1$ , then the procedure is obvious, so assume  $M \geq 2$ .

Let  $X = X_1, X_2, \dots, X_p$  be the corresponding sequence of period lengths, and define  $T(i)$  as before.

We may enumerate the  $\sigma$  in the following way:<sup>8</sup>

Let  $i = 1$  and choose the first  $k$  such  $M \leq \sum_{i \leq j \leq k} X_j$ . That is, choose the first  $k$  for which the sum of the  $X_i$ s is at least  $M$ . Then the basic inequalities hold, so

$$\sigma_1 = X_1, \dots, X_k$$

is a basic subsequence of periods. Compute  $|\delta_\sigma|$  for this  $\sigma$ , and retain it for comparison with subsequent values of other  $\sigma$  (ditto for  $L_0$  and  $M_0$  to avoid redundant computations).

Suppose  $\sigma_m = X_i, \dots, X_n$  is the  $m$ -th basic subsequence of periods. If  $X_i, \dots, X_n, X_{n+1}$  still satisfies the basic inequalities, then  $\sigma_{m+1} = X_i, \dots, X_n, X_{n+1}$  is its successor in the ordering. Otherwise, set  $i = i + 1$  and find the smallest  $k \geq n$  such that  $X_i, \dots, X_k$  satisfies the basic inequalities. Compute  $|\delta_\sigma|$  for this  $\sigma$  and replace the currently saved value of this quantity with it, if it is smaller. Also, compute  $L_0$  and  $M_0$  for this  $\sigma$ , and retain them for calculating their counterparts in the next  $\sigma$ .

After the last  $\sigma$  has been generated, the currently saved value of  $|\delta_\sigma|$  is the required  $L$ .

### **Example**

In the following 28 day work period, what are the most number of consecutive days off which appear in each 7 day interval?

W = 1 1 1 1 0 0 1 1 1 0 0 1 1 0 0 0 0 1 1 0 0 0 1 1 1 0 0 0  
 N = 28  
 M = 7

First build  $X$ , along with  $T$ , to simplify the problem:

X = 4 2 3 2 2 4 2 3 3 3  
 T = 1 0 1 0 1 0 1 0 1 0

Then begin enumerating the basic subsequence of periods of  $W$ , calculating the best rest for each of them, always keeping the smallest<sup>9</sup>:

$$\begin{aligned} \sigma_1 &= 4, 2, 3 \\ L_0 &= 2 \\ M_0 &= 2 \\ L_1 &= 0 \end{aligned}$$

<sup>8</sup> The ordering can be viewed lexicographically.

<sup>9</sup> Actually,  $X$  can be built iteratively while computing the  $\sigma$ s so that only one pass through  $W$  is required.

$$|\delta_{\sigma_1}| = 2 \quad \leftarrow \quad \text{Smallest } |\delta_{\sigma}| \text{ so far} = 2$$

$$\sigma_2 = 4, 2, 3, 2$$

$$L_0 = 2$$

$$M_0 = 5$$

$$L_1 = 7 - 5 - \min(7 - 5 - 1, 4) = 1$$

$$|\delta_{\sigma_2}| = 2 \quad \leftarrow \quad \text{Smallest } |\delta_{\sigma}| \text{ so far} = 2$$

$$\sigma_3 = 2, 3, 2$$

$$L_0 = 0$$

$$M_0 = 3$$

$$L_1 = 7 - 3 - \min(2, 2, \lfloor (7 - 3)/2 \rfloor) = 2$$

$$|\delta_{\sigma_3}| = 2 \quad \leftarrow \quad \text{Smallest } |\delta_{\sigma}| \text{ so far} = 2$$

$$\sigma_4 = 2, 3, 2, 2$$

$$L_0 = 2$$

$$M_0 = 5$$

$$L_1 = 7 - 5 - \min(7 - 5 - 1, 2) = 1$$

$$|\delta_{\sigma_4}| = 2 \quad \leftarrow \quad \text{Smallest } |\delta_{\sigma}| \text{ so far} = 2$$

$$\sigma_5 = 3, 2, 2$$

$$L_0 = 2$$

$$M_0 = 2$$

$$L_1 = 7 - 2 - \min(7 - 2 - 1, 3) = 2$$

$$|\delta_{\sigma_5}| = 2 \quad \leftarrow \quad \text{Smallest } |\delta_{\sigma}| \text{ so far} = 2$$

$$\sigma_6 = 3, 2, 2, 4$$

$$L_0 = 2$$

$$M_0 = 4$$

$$L_1 = 7 - 4 - \min(7 - 4 - 1, 3) = 1$$

$$|\delta_{\sigma_6}| = 2 \quad \leftarrow \quad \text{Smallest } |\delta_{\sigma}| \text{ so far} = 2$$

$$\sigma_7 = 2, 2, 4$$

$$L_0 = 0$$

$$M_0 = 2$$

$$L_1 = 7 - 2 - \min(2, 4, \lfloor (7 - 2)/2 \rfloor) = 3$$

$$|\delta_{\sigma_7}| = 3 \quad \leftarrow \quad \text{Smallest } |\delta_{\sigma}| \text{ so far} = 2$$

$$\sigma_8 = 2, 4, 2$$

$$L_0 = 4$$

$$M_0 = 4$$

$$L_1 = 0$$

$$|\delta_{\sigma_8}| = 4 \quad \leftarrow \quad \text{Smallest } |\delta_{\sigma}| \text{ so far} = 2$$

$$\sigma_9 = 4, 2, 3$$

$$L_0 = 0$$

$$M_0 = 2$$

$$L_1 = 7 - 2 - \min(4, 3, \lfloor (7 - 2)/2 \rfloor) = 3$$

$$|\delta_{\sigma_9}| = \mathbf{3} \quad \leftarrow \quad \text{Smallest } |\delta_{\sigma}| \text{ so far} = 2$$

$$\sigma_{10} = 4, 2, 3, 3$$

$$L_0 = 3$$

$$M_0 = 5$$

$$L_1 = 7 - 3 - 3 = 1$$

$$|\delta_{\sigma_{10}}| = \mathbf{3} \quad \leftarrow \quad \text{Smallest } |\delta_{\sigma}| \text{ so far} = 2$$

$$\sigma_{11} = 2, 3, 3$$

$$L_0 = 3$$

$$M_0 = 3$$

$$L_1 = 0$$

$$|\delta_{\sigma_{11}}| = \mathbf{3} \quad \leftarrow \quad \text{Smallest } |\delta_{\sigma}| \text{ so far} = 2$$

$$\sigma_{12} = 3, 3, 3$$

$$L_0 = 0$$

$$M_0 = 3$$

$$L_1 = 7 - 3 - \min(3, 3, \lfloor (7 - 3)/2 \rfloor) = 2$$

$$|\delta_{\sigma_{12}}| = \mathbf{2} \quad \leftarrow \quad \text{Smallest } |\delta_{\sigma}| \text{ so far} = 2$$

Therefore, the answer is 2.

In general, this procedure will run fast<sup>10</sup> because:

- It examines fewer than  $N - M$  intervals, required by the naive approach which checks each interval of length  $M$ . In this example, 12 intervals are checked, instead of 21.
- Each checked interval can use information from its predecessor to speed up the computation (eg.  $L_0$ ,  $M_0$ , etc.).
- The computation itself is simple algebra - no “sliding” checks are used.

Important point:

This routine's efficiency is independent of the number of time units, depending only on how many times those units change from rest to work, and vice versa. So if the above calendar were changed to hours, instead of days, it wouldn't effect the run time.

---

<sup>10</sup> In linear time, actually.

## Questions

The above algorithm will quickly find the largest  $L$  such that  $\text{Sliding}(W,N,M,L)$  holds, but it also shows where it fails for anything larger. However, what about finding the minimal changes required to boost the guaranteed rest period? In the above example, the minimum rest numbers for the 12 intervals are:

2, 2, 2, 2, 2, 2, 3, 4, 3, 3, 3, 2.

So changing the last interval by resting an additional day (eg. day 23) would improve the last half of the month overall but obviously have no effect on the first half. I think that this method may allow us to quickly find the smallest number of changes which would improve the whole period at once.

More generally, the method presented here might be used as a basic method for more general rules involving sliding checks.

For example, the original example might have been stated:

*Rosters of one month's duration must have at least  $L = 36$  hours free of duty,  $M = 18$  of them consecutive, among any 7 consecutive days. If not, find the largest  $L$  and  $M$  (in that order) such that the rule holds.*

A general theory is possible for such extensions.

# A Mathematical Model For Describing Preferential Optimization

## Introduction

This is a precise mathematical model defining preferential optimization<sup>11</sup>.

## Definitions

A *preferential space* is any tuple  $\langle I, D, \text{con}, \text{com}, \text{obj} \rangle$  where:

$I$  is a linearly ordered nonempty finite set

$D = \langle D_i, i \in I \rangle$  is a set of partially ordered nonempty disjoint finite sets  $D_i$ , indexed on  $I$

$\text{con}$  is a Boolean valued function on all tuples of the form  $\langle d_i, \dots \rangle$ , where  $d_i \in D_i, \dots$

$\text{com}$  is a Boolean valued function on all tuples of the form  $\langle d_i, \dots \rangle$ , where  $d_i \in D_i, \dots$

$\text{com} \leq \text{con}$

$\text{obj}$  is a real valued function on all tuples of the form  $\langle d_i, \dots \rangle$ , where  $d_i \in D_i, \dots$

Without ambiguity,  $\langle$  will denote any of the above orderings.

Because the sets  $D_i$  are disjoint, it is clear that  $\langle d_i, d_j, \dots \rangle = \langle e_k, e_l, \dots \rangle$  implies  $i = k, j = l, \dots$ . So the notational convention that  $d_i \in D_i$  may be relaxed where convenient. In such cases, however, superscripts will be used instead, with the convention that the  $d$ 's appear in the same order as the indexing set  $I$ .

The set  $D_i$  is called the *i-th domain* of the preferential space. The *index* of  $d_i, d^i$  and  $D_i$  is  $i$ .

Any tuple  $A$  such that  $\text{con}(A) = 1$  is called *consistent*, while  $\text{com}(A)$  means that  $A$  is *complete*.

The inequality  $\text{com} \leq \text{con}$  insures that complete tuples are always consistent.

Domain ordering is extended across domains in the following way<sup>12</sup>: for any pair of domain elements  $d_i$  and  $e_j$ ,  $d_i < e_j$  if  $i < j$  or  $i = j$  and  $d_i < e_j$ .

Clearly the union  $\cup D_i$  of the  $D_i$ s under  $\langle$  is a partial order, and if all the  $D_i$ s are linearly ordered, then so is the union.

<sup>11</sup> A major problem in airline scheduling is first defining the problem.

<sup>12</sup> This definition is unambiguous because the domains are disjoint, by definition.

More generally, tuples inherit this order by comparing the first element where they differ. To define this more precisely, some simple definitions are required. The *length* of the tuple  $\langle d^1, \dots, d^n \rangle$  is  $n$ . It is also a *subsequence* of any tuple of the form  $\langle d^1, \dots, d^n, \dots \rangle$ . If the length of the latter is greater than  $n$ , then the former is a *proper subsequence* of it. Finally,  $\langle d^1, \dots, d^m \rangle < \langle e^1, \dots, e^n \rangle$  if  $\langle d^1, \dots, d^m \rangle$  is a proper subsequence of  $\langle e^1, \dots, e^n \rangle$  or, for some  $p > 0$ ,  $d^p < e^p$  and  $d^i = e^i$  for  $i < p$ .

Clearly the set of all tuples under  $<$  is a partial order, and if all the  $D_i$ s are linearly ordered, then so is the set of all tuples.

*From now on, unless otherwise specified, all the  $D_i$ s are assumed to be linearly ordered.*

Viewing the  $D_i$ s as sets of letters of an alphabet ordered by  $<$ , where all the letters in  $D_i$  come before those in any  $D_j$  ( $j > i$ ), then the set of all tuples, or words, obeys standard dictionary order (ie. lexicographic order).

If  $S$  is an arbitrary set of tuples,  $s \in S$ , and  $\text{obj}(s) \geq \text{obj}(s')$  for all  $s' \in S$ , then  $s$  is an *optimal tuple* of  $S$  with respect to  $\text{obj}$ . If  $s' > s$  for any other optimal tuple  $s'$ , then  $s$  is the *most preferred tuple* of  $S$  with respect to  $\text{obj}$ . Clearly there is exactly one most preferred tuple. The  *$i$ -th most preferred tuple* may be defined by induction in the obvious way<sup>13</sup>, and there is one such tuple for every  $i$ . Therefore, any set  $S = \{s_1, s_2, \dots\}$  has a unique permutation  $p$  such that  $\{s_{p(1)}, s_{p(2)}, \dots\}$  lists its elements in increasing order of preference, ie.  $s_{p(i)}$  is the  $i$ -th most preferred tuple, for  $i = 1, 2, \dots$ . This permutation is called the *preferred permutation* for  $S$ , with respect to  $\text{obj}$ .

The basic problem in preferential optimization is to find the most preferred tuple among all the complete tuples, with respect to the objective function.

Another is to find the  $i$ -th most preferred tuple among all the complete tuples, with respect to the objective function.

If there are no complete tuples, then a basic problem is to find the most preferred tuple among all the consistent tuples, with respect to the objective function, or else find the  $i$ -th most preferred tuple.

Note that if the objective function is a constant over any set, then its preferred permutation simply lists its elements in dictionary order.

### **Example 1 (Preferential optimization for individual crew member)**

Suppose you want to schedule a crew member with minimum credit, providing that a legal assignment of pairings can be found within a fixed credit window. Otherwise, assign a legal set of pairings which maximizes the credit, providing it is below the credit window.

<sup>13</sup> Eg. After removing the first most preferred element  $s$  from  $S$ , the second most preferred element is the first most preferred element of  $S - \{s\}$ .

$I$  = set of bids ordered by preference

$D_i$  = set of feasible assignments for the  $i$ -th bid

$con(\langle d^1, \dots, d^i, \dots \rangle)$  = 1 if  $d^1, \dots, d^i, \dots$  can be legally assigned to the above crew member and the total pairing credit is not above the credit window  
= 0, otherwise

$com(\langle d^1, \dots, d^i, \dots \rangle)$  = 1 if  $con(\langle d^1, \dots, d^i, \dots \rangle)$  is 1 and the total pairing credit is within the credit window  
= 0, otherwise

$obj(\langle d^1, \dots, d^i, \dots \rangle)$  = negative of the total pairing credit if  $com(\langle d^1, \dots, d^i, \dots \rangle)$  is 1  
= total pairing credit, otherwise

Each element of  $D_i$  is a legal set of pairings for the crew member, ordered by preference. Whether any collection  $S$  of them can be legally assigned together is determined by  $con(S)$ . Furthermore,  $com(S)$  determines if a legal assignment  $S$  completes the schedule.

Then the problem is to find the most preferred element among all the complete tuples, with respect to the objective function.

If there are no complete tuples, then find the most preferred element among all the consistent tuples, with respect to the objective function.

If there are no consistent tuples, then the problem is insoluble.

## **Example 2 ( Preferential optimization for all crew members)**

Suppose you want to schedule all crew members in order of seniority, until all of the pairings have been assigned to crew. No crew member is assigned unless a complete schedule is available, and the problem is considered insoluble unless all pairings have been assigned.

$I$  = set of crew ordered by seniority

$D_i$  = list of all possible complete assignments for the  $i$ -th crew member, ordered by preference

$con(\langle d^1, \dots, d^i, \dots \rangle)$  = 1 if  $d^1, \dots, d^i, \dots$  can be legally assigned to corresponding crew members and hard stacking days are covered  
= 0, otherwise

$com(\langle d^1, \dots, d^i, \dots \rangle)$  = 1 if  $con(\langle d^1, \dots, d^i, \dots \rangle)$  is 1 and all pairings have been assigned to crew  
= 0, otherwise

$obj(\langle d^1, \dots, d^i, \dots \rangle)$  = 1

Then the problem is to find the most preferred element among all the complete tuples. If there are no complete tuples, then the problem is insoluble (unless some further optimization is desired among the consistent tuples).

### **Important point**

Each  $D_i$  represents all possible complete assignments for the  $i$ -th crew member, not just the ones that are available when scheduling begins for that crew. That's because we're stating what the best answer is, not how to compute it, which is a different problem.

### **Note**

In Example 2, we could vary the problem and allow for partial scheduling of crew members by changing the con/com/obj functions for each crew.

Or, independently, we could allow for partial scheduling of the pairings by doing the same for the con/com/obj functions on all the crew.

The point is that all these variants on what preferential optimization means are defined as instances of the general problem. In each case we have an exact definition of what we're trying to achieve, once everyone agrees on what the con/com/obj functions are.

### **Implementation problems and opportunities**

Although preferential optimization has been stated in the language of constraint satisfaction, that may not be the most efficient paradigm since lookahead and truth maintenance techniques are resource-hungry. But some of the tree paring algorithms found within that approach may be useful, even for large problems. In any event, we can now relate each technique directly back into our own problems whenever they adhere to the preferential optimization template.

### **Generalization to buddy bids**

The following generalization of preferential spaces is designed to handle "buddy" bids, where one crew member specifically requests assignments with other, more senior, members. Consequently, which assignments are preferred depend on the senior crew members' current assignments so we need some way of reordering a given crew member's preferences during scheduling, while still preserving the definition of "first" choice among many.

### **Definition**

A *generalized preferential space* is any preferential space  $\langle I, D, \text{con}, \text{com}, \text{obj} \rangle$  where the order on each  $D_i$  is replaced by a set of orders, one for each tuple  $\langle d_j, \dots, d_k \rangle$  in which  $k < i$ <sup>14</sup>.

---

<sup>14</sup> Including the empty tuple  $\langle \rangle$ . If all tuples generate the same ordering as  $\langle \rangle$ , then a generalized preferential space is just an ordinary preferential space.

## **Comment**

The motivation behind this is the following:

Once you've selected any elements  $\langle d_j, \dots, d_k \rangle$  from any set of domains preceding the  $i$ -th domain, the order of that domain is then determined. In particular, selecting a different set of elements might change the order. However, any selection from those domains equal to or succeeding the  $i$ -th domain would have no effect on the  $i$ -th domain.

If  $\langle d_j, \dots, d_k \rangle$  represents assignments for crews  $j, \dots, k$  and crew  $i$  wanted to fly with at least one of them, then the order of the  $i$ -th domain would be changed to reflect those pairings which are now assigned to crew  $j, \dots, k$ . This is where generalized preferential spaces are needed. However, we need to show that the original definition of preferred permutation is still meaningful.

## **Definition**

Domain ordering is extended across domains in the same way as before, except that there is a different one for each tuple. Again, the union  $\cup D_i$  of the  $D_i$ s under any such ordering is a partial order, and if all the  $D_i$ s are linearly ordered, then so is the union. For any tuple  $t$ , its partial order on  $\cup D_i$  is sometimes denoted  $<_t$ .

Tuples are ordered by comparing the first element where they differ, namely  $\langle d^1, \dots, d^m \rangle < \langle e^1, \dots, e^n \rangle$  if  $\langle d^1, \dots, d^m \rangle$  is a proper subsequence of  $\langle e^1, \dots, e^n \rangle$  or, for some  $p > 0$ ,  $d^p <_t e^p$  and  $d^i = e^i$  for  $i < p$ , where  $t = \langle d^1, \dots, d^{p-1} \rangle (= \langle e^1, \dots, e^{p-1} \rangle)$ .

It is not immediately obvious that  $<$  is a partial order on all tuples, since the ordering  $<_t$  in the comparison  $d^p <_t e^p$  depends on the elements  $d^i (= e^i)$  for  $i < p$ . So the following theorem is required.

## **Theorem**

The set of all tuples under  $<$  is a partial order.

## **Proof**

Clearly  $<$  is anti-reflexive. So we only need to prove transitivity, ie. for all tuples  $X, Y$  and  $Z$ :

$$X < Y \text{ and } Y < Z \rightarrow X < Z$$

Assume that  $X < Y$  and  $Y < Z$ . We need to prove that  $X < Z$ .

For any pair of tuples  $A, B$  let  $A \subset B$  mean that  $A$  is a subsequence of  $B$ . Clearly  $\subset$  is transitive.

There are four cases to consider, which exhaust all possibilities:

- (1)  $X \subset Z$
- (2)  $X \subset Y$  and  $Y \not\subset Z$  and  $X \not\subset Z$
- (3)  $X \not\subset Y$  and  $Y \subset Z$  and  $X \not\subset Z$
- (4)  $X \not\subset Y$  and  $Y \not\subset Z$  and  $X \not\subset Z$

If (1) holds then  $X < Z$  by definition.

Suppose (2) holds. Then since  $X \subset Y$  and  $X \not\subset Z$ , the first index where  $X$  and  $Z$  are different is the same first index where  $Y$  and  $Z$  are different. So  $X < Z$  and  $Y < Z$  are both determined with the same domain elements  $x^p = y^p$  and  $z^p$  using the same order  $<_t$ , where  $t$  is the maximum subsequence common to  $X$  and  $Z$  (which is the same maximum subsequence common to  $Y$  and  $Z$ ). Since  $y^p <_t z^p$  and  $x^p = y^p$ , the result follows.

Suppose (3) holds. Then since  $X \not\subset Y$  and  $Y \subset Z$  the first index where  $X$  and  $Y$  are different is the same first index where  $X$  and  $Z$  are different. So  $X < Y$  and  $X < Z$  are both determined with the same domain elements  $x^p$  and  $y^p = z^p$  using the same order  $<_t$ , where  $t$  is the maximum subsequence common to  $X$  and  $Y$  (which is the same maximum subsequence common to  $X$  and  $Z$ ). Since  $x^p <_t y^p$  and  $y^p = z^p$ , the result follows.

Suppose (4) holds. Then there are three possibilities:

- (4.1) The first index where  $X$  and  $Y$  are different is the same as the first index where  $Y$  and  $Z$  are different
- (4.2) The first index where  $X$  and  $Y$  are different precedes the first index where  $Y$  and  $Z$  are different
- (4.3) The first index where  $X$  and  $Y$  are different succeeds the first index where  $Y$  and  $Z$  are different

Suppose (4.1) holds. Then  $X < Y$ ,  $Y < Z$  and  $X < Z$  are all determined with distinct domain elements  $x^p$ ,  $y^p$ ,  $z^p$  using the same order  $<_t$ , where  $t$  is the maximum subsequence common to  $X$ ,  $Y$ , and  $Z$ . By transitivity on  $<_t$ , the result follows.

Suppose (4.2) holds. Let  $x^p$  and  $y^p$  be the elements of  $X$  and  $Y$  corresponding to the first index  $p$  where they differ. Because this index precedes the first index where  $Y$  and  $Z$  differ,  $x^p$  and  $y^p$  must also be the elements of  $X$  and  $Z$  corresponding to the first index where they differ. So  $X < Z$  for the same reason that  $X < Y$ .

Suppose (4.3) holds. Let  $y^p$  and  $z^p$  be the elements of  $Y$  and  $Z$  corresponding to the first index  $p$  where they differ. Because this index precedes the first index where  $X$  and  $Y$  differ,  $y^p$  and  $z^p$  must also be the elements of  $X$  and  $Z$  corresponding to the first index where they differ. So  $X < Z$  for the same reason that  $Y < Z$ .

### **Corollary**

The ordering  $<$  on all tuples is linear, if all the  $D_i$ s are linearly ordered.

### **Proof**

Immediate.

# Using ILOG and Hall's Theorem

## Introduction

This note shows that the common constraints of airline stacking<sup>15</sup> and allocating the correct number of preferred pairings<sup>16</sup> to selected crew<sup>17</sup> can be handled by the same computational vehicle. In particular, both constraints may be viewed as special cases of Hall's theorem<sup>18</sup> for choosing distinct representatives from finite sets. Implementing any generalization of these constraints would follow from a corresponding generalization of Hall's theorem.

## Theorem (Hall)

Given any finite collection of finite sets  $S_i$ , there exists distinct elements  $s_i \in S_i$  if and only (iff)  $|\bigcup_{j \in J} S_j| \geq |J|$  for every  $J \subseteq I$ .

## Remark

Hall's theorem states that if you can select elements from each  $S_i$  which are different for each  $i$ , then whenever you collect together some of those sets, say 10 of them, their union must contain at least as many elements as the number of sets you collected together (in this case, 10). That this is necessary is obvious, since the union must contain the  $s_i$  chosen from each  $S_i$ , and they are distinct by hypothesis. The hard part is showing that the reverse is true, yielding a computation for when such a "system of distinct representatives" is possible.

For example, let:

$$\begin{aligned} S_1 &= \{1\} \\ S_2 &= \{3,4\} \\ S_3 &= \{1,4\} \\ S_4 &= \{1,2\} \end{aligned}$$

Can distinct representatives be found? By inspection, yes:

$$\begin{aligned} s_1 &= 1 \\ s_2 &= 3 \\ s_3 &= 4 \\ s_4 &= 2 \end{aligned}$$

But Hall's theorem guarantees such a selection, since:

---

<sup>15</sup> Those pairings passing through any day must be assigned different crew.

<sup>16</sup> A pairing is any sequence of flights starting and ending at the same crew base

<sup>17</sup> Each crew must be assigned the same number of desirable pairings as others.

<sup>18</sup> Hall, P. "On representatives of subsets", J. London Math. Soc., 10 (1935), 26-30

Any set contains at least 1 element:

$$|S_1| = |\{1\}| = 1 \geq 1$$

$$|S_2| = |\{3,4\}| = 2 \geq 1$$

$$|S_3| = |\{1,4\}| = 2 \geq 1$$

$$|S_4| = |\{1,2\}| = 2 \geq 1$$

Any two sets contains at least 2 elements:

$$|S_1 \cup S_2| = |\{1,3,4\}| = 3 \geq 2$$

$$|S_1 \cup S_3| = |\{1,4\}| = 2 \geq 2$$

$$|S_1 \cup S_4| = |\{1,2\}| = 2 \geq 2$$

$$|S_2 \cup S_3| = |\{1,3,4\}| = 3 \geq 2$$

$$|S_2 \cup S_4| = |\{1,2,3,4\}| = 4 \geq 2$$

$$|S_3 \cup S_4| = |\{1,2,4\}| = 3 \geq 2$$

Any three sets contains at least 3 elements:

$$|S_1 \cup S_2 \cup S_3| = |\{1,3,4\}| = 3 \geq 3$$

$$|S_2 \cup S_3 \cup S_4| = |\{1,2,3,4\}| = 4 \geq 3$$

etc.

Any four sets contains at least 4 elements:

$$|S_1 \cup S_2 \cup S_3 \cup S_4| = |\{1,2,3,4\}| = 4 \geq 4$$

On the other hand, if  $S_2$  were changed to  $\{1,4\}$  then

$$|S_1 \cup S_2 \cup S_3| = |\{1,4\}| = 2 \geq 3 \text{ FALSE}$$

so the theorem would fail, and a system of distinct representatives wouldn't be possible.

Actually, Hall's theorem can be generalized in the following way:

### **Corollary**

Given any finite collection of finite sets  $S_i$ , and arbitrary numbers  $n_i$ , there exist disjoint subsets  $s_i \subseteq S_i$  containing exactly  $n_i$  elements iff  $|\bigcup_{j \in J} S_j| \geq \sum_{j \in J} n_j$  for every  $J \subseteq I$ .

In other words, we can select predetermined numbers of elements from each  $S_i$  (where no element is selected twice) iff whenever you collect together some of those sets their union contains at least as many elements as the number of elements you collectively want to select from them.

## Remark

In this memo, both Hall's theorem and its corollary will be referred to as Hall's result, without confusion.

## Example

### *1. Input data:*

#### Pairings:

	Day 1	Day 2	Day 3
Pairing 1	O-----O		
Pairing 2	O-----O		
Pairing 3		O	
Pairing 4		O	
Pairing 5			O
Pairing 6			O
Pairing 7			O

#### Crew Availabilities:

Crew 1 is available for Pairings 1,2,3,4

Crew 2 is available for Pairings 1,2,3,4,5,6

Crew 3 is available for Pairings 3,4,5,6,7

Crew 4 is available for Pairings 3,4,7

#### Preferred pairings to selected crew:

The preferred pairings are 3,4,5,6. Crew 2, 3, 4 must have one, two, one preferred pairings, respectively.

## 2. Representing the input data using Boolean matrices

The following Crew Availability Day (CAD) matrix will be used to summarize the problem:

Day			Pairing	Crew/Number				Preferred	Pairings
1	2	3		1	2	3	4		
					1	2	1	←----- *	
				-----					
1	1		1	1	1	0	0		
1	1	1	2	1	1	0	0	↓	
	1		3	1	1	1	1	1	
	1		4	1	1	1	1	1	
		1	5	0	1	1	0	1	
		1	6	0	1	1	0	1	
		1	7	0	0	1	1		
				-----					

That is,

Stacking:

Day 1 encounters Pairings 1,2

Day 2 encounters Pairings 1,2,3,4

Day 3 encounters Pairings 2,5,6,7

Crew availability:

Crew 1 is available for Pairings 1,2,3,4

Crew 2 is available for Pairings 1,2,3,4,5,6,7

Crew 3 is available for Pairings 3,4,5,6,7

Crew 4 is available for Pairings 3,4,7

Pairing preferences:

Crew 2 requires one of Pairings 3,4,5,6

Crew 3 requires two of Pairings 3,4,5,6

Crew 4 requires one of Pairings 3,4,5,6

## 3. The basic problem

The basic problem is to select a single 1 from each row between the dotted lines so that on each day no crew member flies more than one pairing (stacking) and each crew member is assigned the indicated number of preferred pairings (pairing preferences).<sup>19</sup>

Note that the given pairing preferences are completely defined by the single row and column denoted by \*, but that other such rows and columns could be present at the same time (using different crew and pairing combinations).

#### 4. Describing stacking constraints in the language of Hall's theorem

In order to show that the stacking constraints are special cases of Hall's theorem, we need to specify some important sets.

Let  $S$  be the set of all pairings:

$$S = \{1,2,3,4,5,6,7\}$$

The stacking columns effectively represent various subsets of  $S$  (ie. wherever the 1's occur). In particular, the stacking sets  $S_i$  are:

$$S_1 = \{1,2\}$$

$$S_2 = \{1,2,3,4\}$$

$$S_3 = \{2,5,6,7\}$$

First note that  $S_1$  can be discarded, since it is a subset of  $S_2$  and yields no constraint not already implied by the bigger set. In other words,  $S_2$  already tells us that the first two pairings must be assigned to different crew. More generally, only those stacking sets not properly contained in other stacking sets need be retained. Such sets, called *maximal stacking sets*, are pre-selected before proceeding further.<sup>20</sup> We will now assume that this has been done, so our stacking sets  $S_i$  are now:

$$S_2 = \{1,2,3,4\}$$

$$S_3 = \{2,5,6,7\}$$

An algorithm for selecting maximal stacking sets is given in the Appendix, and need not be presented here.

Each of these stacking sets determines various subsets of the crew  $C = \{1,2,3,4\}$  in the following way:

Take  $S_2$ . Each element  $i$  of  $S_2$  represents a row of 1s which effectively selects a subset  $C_i$  of  $C$ , namely:

$$C_1 = \{1,2\}$$

$$C_2 = \{1,2\}$$

$$C_3 = \{1,2,3,4\}$$

$$C_4 = \{1,2,3,4\}$$

The stacking constraint represented by  $S_2$  is equivalent to the following:

---

<sup>20</sup> Strictly speaking, non-maximal sets are deleted only to avoid redundant computations.

Select a different element from each  $C_i$

But this is handled by Hall's theorem. In particular, the constraint may be satisfied iff the following hold:

$|C_1| \geq 1$ , ie.  $C_1$  contains at least one element

$|C_2| \geq 1$

$|C_3| \geq 1$

$|C_4| \geq 1$

$|C_1 \cup C_2| \geq 2$ , ie.  $C_{21}$  and  $C_{22}$  collectively contain at least two elements

$|C_1 \cup C_3| \geq 2$

$|C_1 \cup C_4| \geq 2$

$|C_2 \cup C_3| \geq 2$

$|C_2 \cup C_4| \geq 2$

$|C_3 \cup C_4| \geq 2$

$|C_1 \cup C_2 \cup C_3| \geq 3$  etc.

The stacking constraint represented by  $S_3$  is handled by a different instance of the same theorem, analogously.

So verifying whether each constraint is satisfiable, independently of the others, is merely a matter of computing Hall's theorem for each maximal stacking set. Of course, satisfying them all simultaneously is another matter, since one implies the other but not the other way around.

##### *5. Describing pairing preference constraints in the language of Hall's theorem*

In order to show that the pairing preference constraints are special cases of Hall's theorem, we need to specify the sets involved.

The base set is simply the set of all preferred pairings, denoted by the preference column:

$P = \{3,4,5,6\}$

The availability matrix restricted to these pairings denotes the preferred and available sets  $P_i$  for each crew member:

$P_1 = \{3,4\}$

$P_2 = \{3,4,5,6\}$

$P_3 = \{3,4,5,6\}$

$P_4 = \{3,4\}$

The preference row tells us how many elements from each  $P_i$  are required to satisfy the constraint:

- 0 elements must be chosen from  $P_1$
- 1 elements must be chosen from  $P_2$
- 2 elements must be chosen from  $P_3$
- 1 elements must be chosen from  $P_4$

Of course, the elements chosen from each  $P_i$  must be disjoint from the others, since a pairing cannot be assigned to two crew simultaneously. Again, this is exactly handled by Hall's theorem. In particular, an assignment is possible for this constraint, independent of the others, iff:

$$|P_1 \cup P_2| \geq 0 + 1$$

$$|P_1 \cup P_3| \geq 0 + 2$$

$$|P_1 \cup P_4| \geq 0 + 1$$

$$|P_2 \cup P_3| \geq 1 + 2$$

$$|P_2 \cup P_4| \geq 1 + 1$$

$$|P_3 \cup P_4| \geq 2 + 1$$

$$|P_1 \cup P_2 \cup P_3| \geq 0 + 1 + 2$$

etc.

$$|P_1 \cup P_2 \cup P_3 \cup P_4| \geq 0 + 1 + 2 + 1$$

## **Summary**

We have shown that any constraint which effectively implies that different pairings must have different crew, or selected crew must have predetermined numbers of selected pairings, can be tested with Hall's theorem. All such constraints are basically equivalent in nature.

But suppose we wanted to test more general constraints, such as bounding the number of selected pairings that selected crew might be assigned, rather than fixing the exact number.

Then we might like a generalization of Hall, using matching results in the literature.

Note that ILOG's library naturally encodes Hall's theorem, since you can set up pointers to any sets of variable domains and insist that selected elements always be distinct.

## **Appendix**

Problem:

Given stacking sets  $S_i$ , erase those properly contained in the others

## **Algorithm:**

1. Pick the first set
2. Look for a later set containing it, ie. equal to or bigger than it. Go to 6 if there are no later sets.
3. As soon as one is found, erase the original set, pick the next set and go to 2. Go to 6 if no next set remains.
4. While searching, erase any set it contains, ie. equal to or smaller than it
5. If none is found in 3, pick the next set and go to 2. Go to 6 if there are no next sets.
6. The remaining sets now form a maximal collection (under inclusion)