On Improving Hospital Service for Americans and Russians

LOGAN WU
ANDREW JACKSON
SCOTT SUNG

Department of Engineering Science

SEPTEMBER 28, 2017 The University of Auckland

Abstract

In this report, we describe the mixed-integer formulation to balance patient numbers at a hospital by rostering the registrars in each ward. We predict the infinite-horizon mean occupancies of each ward assuming a Poisson distribution of discharges.

We estimate that our program can reduce the average daily imbalance (the difference in patients between the most and least occupied wards each day) by approximately 23% compared to a random, unoptimised roster. A pairwise analysis of the results reveals that even in the scenario where registrars cannot change their starting weeks, the optimisation program is still able to significantly reduce imbalances.

1 Introduction

This linear program generates feasible rostering solutions with the aim of minimising the expected disparity between the occupancies of the hospital's wards. The program is flexible, easily accommodating different staff arrangements and historical data with small modifications to a data file.

2 Assumptions

2.a Independent, Identically Distributed People

To calculate the outcome of a roster, we take advantage of symmetry in the problem and make assumptions about the lengths of stays.

We assume that all ward teams act equally and independently, i.e. there is no difference if we swap the Navy and Lime teams. This is a fair assumption because registrars usually all have the same background and ward teams are physically separated.

A trickier assumption is that all patients are independent and memoryless. This allows us to model them with a Poisson distribution where the mean stay is 2.1 days (Simulation Project, ENGSCI 355). In reality this should be a heavy-tail distribution (if a patient is going to stay for a while, they will be there for a *long* time), but the memoryless property is a powerful approximation. It also makes the hospital stable, so over a long period of time the number of discharges converges to the number of admissions (everyone is eventually accounted for).

2.b Random Admissions & Lengths of Stays

We assume that there is no trend in the mean admissions between weeks – even though there will be random fluctuations, numbers will be centered on a mean. This allows us to model admissions as an average of each weekday for the past 15 weeks.

If there is no trend in the length of stay, we can model discharges with a probability distribution. This might be violated, for example, if doctors get faster at treating patients with more experience.

3 Objective

We have defined our objective as minimising the average expected disparity between the most and least occupied wards. We average over a looping, 42-day cycle. This estimate gives us a statistical average for the overall perceived difference in ward occupant numbers. We choose to minimise the average over the whole roster rather than the peaks for two reasons:

1. This objective drives the program to minimise the difference over every single day, rather than focusing only on peak differences; the hospital should notice a constant improvement.
2. Peaks are far more difficult to control and cannot be reliably reduced ahead of time due to their random nature.

4 Results

By comparing a series of random, unoptimised rosters ($N = 30$, obtained by disabling the objective function) with our roster (Table 1), we observe a significant improvement in the daily ward imbalances (Figure 1). On a day-to-day basis using a random roster, the most occupied ward will have an average of between 34 and 36 people ($\alpha = 0.05$) more people than the least occupied. After optimisation, this decreased to 27 – an estimated reduction of 8 people or 23%.

While investigating the relative effects, we found no evidence that the 1&4, 2&5, 3&6 week arrangement

Table 1: Final optimised roster. The colour codes are the initial starting weeks for each registrar. Notably, each ward has its registrars three weeks apart.

Week	Mon	Tue	Wed	Thu	Fri	Sat	Sun
$\bullet_1$ 1	O	O	O	O	O	A	A
$\bullet_1$ 2	P	O	O	A	P	X	X
$\bullet_1$ 3	O	O	O	O	A	P	X
$\bullet_2$ 4	A	P	A	P	O	X	X
$\bullet_2$ 5	O	O	O	O	N	N	N
$\bullet_2$ 6	NO	NA	NP	NO	ZO	ZX	ZX

was significant. However, a pairwise analysis before and after adding the objective function (each pair with the same starting week arrangement) found a difference of between 5 and 7 fewer people per day. This shows that even if the starting weeks are fixed, optimisation still provides a significant improvement.

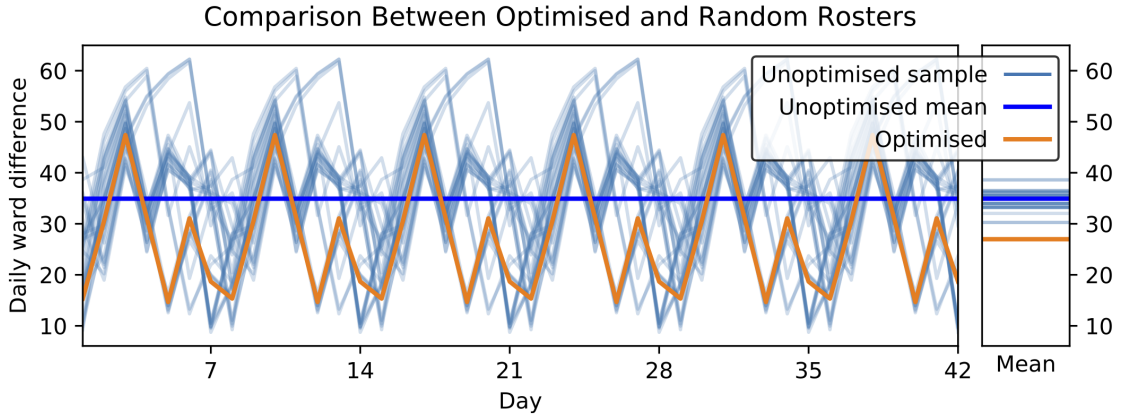

Figure 1: Effect of optimisation on day-to-day patient imbalances

5 Conclusion

Our program significantly reduces hospital ward imbalances compared to a random roster by between 7 and 9 people on average. This results in improved patient care as registrars are more evenly distributed between patients.

Because we are using predictions for the expected patient numbers, our method results in a constant, average improvement every day; not only will the hospital notice a general improvement, peak discrepancies will also be lower than before.

A Formulation

Our model can be conceptualised as two coupled linear programs. The first is a roster of shifts to ensure feasibility, and the second one is a cyclical network representing ward occupancies during the 42 days of the roster.

A.a Parameters

$\mathbb{S} = \{\text{A, P, N, Z, X, O}\}$ ..... Set of shift types
$\mathbb{W} = \{1, \ldots, 6\}$ .... Set of weeks in the roster cycle
$\mathbb{D} = \{\text{Mon}, \ldots, \text{Sun}\}$ ..... Set of days in a week
$\mathbb{Y} = \{0, 0, \ldots, 0, 1\}$ .... Night shift on the last week

For continuity, a dummy week 0 and dummy day $\text{Sun}_{\text{dummy}}$ before Monday also exist but are not part of the sets $\mathbb{W}$ and $\mathbb{D}$. Also, fixing the night shift week takes advantage of symmetry between the weeks.

A.b Decision Variables

X is the array of binary variables determining if a type of shift belongs to a given week and day in the roster:

$$x_{s,w,d} \in \{0,1\} \quad \forall s \in \mathbb{S},\ w \in \mathbb{W} \cup \{0\},\ d \in \mathbb{D} \cup \{\text{Sun}_{\text{dummy}}\}$$

V denotes whether registrars are forced to take a weekend off:

$$v_w \in \{0,1\} \quad \forall w \in \mathbb{W} \cup \{\text{Sun}_{\text{dummy}}\}$$

A.c Constraints

Create a dummy week 0 that is equal to the final week.

$$x_{s,0,d} = x_{s,|\mathbb{W}|,d} \quad \forall s \in \mathbb{S},\ d \in \mathbb{D} \tag{1}$$

Create a dummy day 0 that is equal to the last day of the previous week, allowing wrap-around from Sunday to Monday.

$$x_{s,w-1,|\mathbb{D}|} = x_{s,w,0} \quad \forall s \in \mathbb{S},\ w \in \mathbb{W} \tag{2}$$

Ensure every slot has a shift assigned by summing over all shift types, except for the night-shift week which must have two.

$$\sum_{s \in \mathbb{S}} x_{s,w,d} - y_w = 1 \quad \forall w \in \mathbb{W},\ d \in \mathbb{D} \tag{3}$$

Every day must have a single registrar assigned to each A, P and N shift.

$$\sum_{w \in \mathbb{W}} x_{s,w,d} = 1 \quad \forall s \in \{\text{A, P, N}\},\ d \in \mathbb{D} \tag{4}$$

Every P shift must follow an A shift, except for Sunday where an A must follow an A.

$$x_{\text{P},w,d} = x_{\text{A},w,d-1} \quad \forall w \in \mathbb{W},\ d \in \mathbb{D} \tag{5}$$

$$x_{\text{A},w,d} = x_{\text{A},w,d-1} \quad \forall w \in \mathbb{W},\ d \in \{\text{Sun}\} \tag{6}$$

The Friday to Sunday before the full night shift week are also night shifts.

$$x_{\text{N},w-1,d} - y_w = 0 \quad \forall w \in \mathbb{W},\ d \in \{\text{Fri, Sat, Sun}\} \tag{7}$$

The Monday to Thursday of the full night shift week are night shifts.

$$x_{\text{N},w,d} - y_w = 0 \quad \forall w \in \mathbb{W},\ d \in \{\text{Mon, Tue, Wed, Thu}\} \tag{8}$$

The Friday to Sunday of the full night shift week are sleep shifts.

$$x_{\mathrm{Z},w,d} - y_w = 0 \quad \forall w \in \mathbb{W},\ d \in \{\text{Fri, Sat, Sun}\} \tag{9}$$

Ensure no one else has a sleep shift and is slacking off!

$$\sum_{w \in W,\ d \in \mathbb{D}} x_{\mathrm{Z},w,d} = 3 \text{ (number of allowed rests)} \tag{10}$$

No weekdays are allowed to be taken off.

$$x_{\mathrm{X},w,d} = 0 \quad \forall w \in \mathbb{W},\ d \in \{\text{Mon}, \ldots, \text{Fri}\} \tag{11}$$

Weekends must be taken off if scheduled as a 'weekend off' (but may be taken off on other weekends).

$$x_{\mathrm{X},w,d} \geq 0 \quad \forall w \in \mathbb{W},\ d \in \{\text{Sat, Sun}\} \tag{12}$$

No two consecutive weekends can pass without a weekend off being forced.

$$v_w + v_{w-1} \geq 1 \quad \forall w \in \mathbb{W} \tag{13}$$

A.d Objective Parameters

There are several new definitions that are particular to this section of the formulation. This section can be represented as a 42-node cycle with discounted arcs between consecutive nodes, and artificial insertions on admission days averaged from the 15-week data.

$\mathbb{WA} = \{\text{lime, navy, yellow}\}$ ..... Set of wards
$\mathbb{R} = \mathbb{W} \times \mathbb{D}$ Set of all days in the roster, to simplify $r+1$ and $r-1$ subscripts
$\rho = 0.45$ ..... Poisson constant for discharges
$s_{wa,re}$ ..... See following paragraph

$s_{wa,re} \in \mathbb{W}$ is an array of unique starting weeks for each registrar in each ward. $s_{wa,re}$ is not a decision variable of the IP; rather, it is a meta-variable, enumerated by a tree with 15 unique combinations.

A.e Objective-Related Variables

A is a unimodular matrix determining whether a ward is admitting on a certain day.

$$a_{wa,r} \in \{0,\ 1\} \quad \forall wa \in \mathbb{WA},\ r \in \mathbb{R}$$

O is a matrix of each ward's *expected* occupancy for each day in the roster (we use expectation values to deal with the stochastic nature of ward numbers).

$$o_{wa,r} \geq 0 \quad \forall wa \in \mathbb{WA},\ r \in \mathbb{R}$$

$\dot{\mathrm{P}}$ is the expected daily admission rate for each day of the week.

$$\dot{p_d} \in Z^+ \quad \forall d \in \mathbb{D}$$

Δ is the maximum pairwise difference between all wards for a given day of the roster.

$$\delta_r \geq 0 \quad \forall r \in \mathbb{R}$$

A.f Objective Constraints and Objective Function

These constraints calculate the expected mean occupancy of each ward throughout the 42-day roster. % is the modulo function and $\lceil x \rceil$ rounds up to the nearest integer.

Control whether a ward is admitting on a given day of the 6-week cycle. The absolutely insane subscripts adjust for starting week offsets; i.e. on week one, day one, a registrar who started on week six will actually be looking at week six, day one of the roster.

$$a_{wa,(r+1)\%|\mathbb{R}|+1} = \sum_{re \in \{1 \ldots n_{\text{registrars}}\}} x_{\text{A},\lceil \frac{r+|\mathbb{D}|s_{wa,re}-1}{|\mathbb{D}|} \rceil \%|\mathbb{W}|+1,(r-1)\%|\mathbb{D}|+1} \quad \forall wa \in \mathbb{WA},\ r \in \mathbb{R} \tag{14}$$

Calculate the expected occupancy based on the discounted occupancy from the previous day, plus the previous day's admissions if the ward was admitting. We used a mean stay of 2.1 days to model discharges as a Poisson process with a rate of 0.45.

$$o_{wa,r\%|\mathbb{R}|+1} = o_{wa,r} \times (1-\rho) + a_{wa,r} \times \dot{p}_{(r-1)\%|\mathbb{D}|+1} \quad \forall wa \in \mathbb{WA},\ r \in \mathbb{R} \tag{15}$$

For each day calculate the range of ward occupant numbers (the maximum pairwise difference).

$$\delta_r \geq o_{wa,r} - o_{wb,r} \quad \forall r \in \mathbb{R},\ wa \in \mathbb{WA},\ wb \in \mathbb{WA} \tag{16}$$

Objective function: Minimise the range of patient numbers between wards, summing over all days in the roster.

$$\text{minimise} \sum_{r \in \mathbb{R}} \delta_r \tag{17}$$

A trump.mod

```
model;

# ============================================================
# ROSTER DATA
# ============================================================

param nWeeks;
param nAllowedRests;

# shifts
set REQSHIFTS;
set NONREQSHIFTS;
set SHIFTS := REQSHIFTS union NONREQSHIFTS;

# weeks
set WEEKS := 1..nWeeks;

# days
set WEEKDAYS;
set WEEKENDS;
set DAYS := WEEKDAYS union WEEKENDS;
set NIGHTSHIFTPREVDAYS;
set NIGHTSHIFTDAYS;
set RESTSHIFTDAYS;

param nightShift {{0} union WEEKS} default 0;   # last element set to 1 in .run

# decision variables - include dummy weeks/days 0
var schedule {SHIFTS, {0} union WEEKS, {0} union DAYS} binary;
var weekendsOff {WEEKS} binary;          # no dummy, uses wraparound 'mod'
    instead

# ============================================================
# FEASIBILITY CONSTRAINTS
# ============================================================

# continuity for dummy weeks/days
s.t. DummyWeek {s in SHIFTS, d in DAYS}:
        schedule[s, 0, d] = schedule[s, card(WEEKS), d];
s.t. DummyDay {s in SHIFTS, w in WEEKS}:
        schedule[s, w-1, card(DAYS)] = schedule[s, w, 0];

# every slot has a shift assigned, except for nightshift week with 2
s.t. AlwaysShift {w in WEEKS, d in DAYS}:
        sum {s in SHIFTS} schedule[s, w, d] - nightShift[w] = 1;

# every day has one A/N except for Sunday
s.t. ReqShift {s in REQSHIFTS, d in DAYS}:
        sum {w in WEEKS} schedule[s, w, d] = 1;

# A followed by P or A on Sunday
s.t. PAfterA {w in WEEKS, d in DAYS diff {7}}:
        schedule['P', w, d] = schedule['A', w, d-1];
s.t. AAfterASunday {w in WEEKS, d in {7}}:
        schedule['A', w, d] = schedule['A', w, d-1];

# create night shift
# Fri - Sun night shift in previous week
s.t. NightShiftPrevWeek {w in WEEKS, d in NIGHTSHIFTPREVDAYS}:
        schedule['N', w-1, d] - nightShift[w] = 0;
# Mon - Thurs night shift
```

```
s.t. NightShiftWeek {w in WEEKS, d in NIGHTSHIFTDAYS}:
        schedule['N', w, d] - nightShift[w] = 0;
# Fri - Sun rest shifts
s.t. RestShiftWeek {w in WEEKS, d in RESTSHIFTDAYS}:
        schedule['Z', w, d] - nightShift[w] = 0;
# only three rests in whole schedule
s.t. RestLimit:
        sum {w in WEEKS, d in DAYS} schedule['Z', w, d] = nAllowedRests;

# weekdays on
s.t. WeekdaysOn{w in WEEKS, d in WEEKDAYS}:
        schedule['X', w, d] = 0;
# weekends can be off
s.t. WeekendsOff{w in WEEKS, d in WEEKENDS}:
        schedule['X', w, d] >= weekendsOff[w];

# no consecutive weekends forced on
s.t. ConsecutiveWeekendsOff{w in WEEKS}:
        weekendsOff[w] + weekendsOff[w mod card(WEEKS) + 1] >= 1;

# ================================================================
# OBJECTIVE CONSTRAINTS
# ================================================================

param totalDays;
param nRegistrars;                                                    # no.
    registrars per ward
param discountRate;
param WEEKLYADMISSIONRATE {1..7};                        # average admissions
    per day
set ROSTERDAYS := 1..totalDays;
set WARDS;
           # LIME NAVY YELLOW
var occupancy {WARDS, ROSTERDAYS};
var admitting {WARDS, ROSTERDAYS} binary;
var wardDiff {ROSTERDAYS};                                          # daily ward
    imbalance
var totalWardDiff;
param startingWeeks {WARDS, 1..nRegistrars};# roster starting week per
    registrar

# calculate occupancy for the next day
# is the ward admitting?
# note: week = ceil(r/card(DAYS))
#               day  = (r-1) mod card(DAYS) + 1
s.t. Admittance {wa in WARDS, r in ROSTERDAYS}:
        admitting[wa, (r+1) mod totalDays + 1] = sum {re in 1..nRegistrars}
            schedule['A', ceil((r+7*(startingWeeks[wa, re]-1))/card(DAYS)) mod
            card(WEEKS)+1, (r-1) mod card(DAYS) + 1];

# calculate occupancy
s.t. Occupancy {wa in WARDS, r in ROSTERDAYS}:
        occupancy[wa, r mod totalDays + 1] = occupancy[wa, r]*(1-discountRate)
            + admitting[wa, r]*WEEKLYADMISSIONRATE[(r-1) mod card(DAYS) + 1];

# calculate ward differences
s.t. WardDifferenceA {r in ROSTERDAYS, wa in WARDS, wb in WARDS}:
        wardDiff[r] >= occupancy[wa, r] - occupancy[wb, r];

s.t. TotalWardDifference:
        totalWardDiff = sum {r in ROSTERDAYS} wardDiff[r];

# ================================================================
```

```
# OBJECTIVE FUNCTION
# ================================================================

minimize ObjectiveOn: sum {r in ROSTERDAYS} wardDiff[r];
minimize ObjectiveOff: 0;
```