AMMM Course Project Report

Nurses in Hospital

Asaf Badouh, Alexander Parunov

December 30, 2017

1 Problem Statement

A public hospital needs to design the working schedule of their nurses. We know for, for each hour h, that at least $demand_h$ nurses should be working. We have available a set of *nNurses* and we need to determine at which hours each nurse should be working. However following constraints must be met:

1. Each nurse should work at least *minHours* hours.
2. Each nurse should work at most *maxHours* hours.
3. Each nurse should work at most *maxConsec* consecutive hours.
4. No nurse can stay at the hospital for more than *maxPresence* hours (e.g. is *maxPresence* is 7, it is OK that a nurse works at 2am and also at 8am, but it not possible that he/she works at 2am and also at 9am).
5. No nurse can rest for more than one consecutive hour (e.g. working at 8am, resting at 9am and 10am, and working again at 11am is not allowed, since there are two consecutive resting hours).

The goal is to determine at which hours each nurse should be working in order to **minimize** the **number of nurses** required and satify all above given constrains.

2 Integer Linear Model

Before defining the model and formal solution, set of parameters and decision variables that are used in model must be defined.

2.1 Parameters

- *nNurses:* Int - number of nurses
- *nHours:* Int - number of hours
- *minHours:* Int - minimum hours each nurse should work
- *maxHours:* Int - maximum hours each nurse can work
- *maxConsec:* Int - maximum consecutive hours each nurse can work
- *maxPresence:* Int - maximum number of hours each nurse can be present
- $demand_h[nNurses]$: Int - demanded number of working nurses at each hour, indexed h

2.2 Decision Variables

- $works_{n,h}[nNurses][nHours]$: Boolean - Nurse n works at hour h
- $WA_{n,h}[nNurses][nHours]$: Boolean - Nurse n works after hour h
- $WB_{n,h}[nNurses][nHours]$: Boolean - Nurse n worked before hour h
- $Rest_{n,h}[nNurses][nHours]$: Boolean - Nurse n rests at hour h. However, nurse should have worked before and should work after the resting hour h. Otherwise it's not considered as resting hour.
- $used_n[nNurses]$: Boolean - Nurse n is working

2.3 Objective Function

After defining all parameters and decision variable we might proceed with definition of objective function and formal constaints. In addition to that,for clarity, we denote set of *nHours* as H, and set of *nNurses* as N. So, since the goal of project is to have as few working nurses as possible satisfying $demand_n[nNurses]$ and all constraints, the objective function is:

$$\min \sum_{n=1}^{N} used_n \tag{1}$$

2.4 Constraints

Solution of problem must respect all constraints given in Section 1. Moreover, ILOG model direcly resembles formally stated constraints. Which means, after formally defining constraints, we may obtain an Integer Linear solution. So constraints are following:

Constraint 1: On each hour h at least $demand_h$ nurses should work.

$$\sum_{n=1}^{N} works_{n,h} \geq demand_h, \forall h \in H \tag{2}$$

Constraint 2: Each nurse n should work at least $minHours$ minimum number of hours.

$$\sum_{h=1}^{H} works_{n,h} \geq used_n \times minHours, \forall n \in N \tag{3}$$

Constraint 3: Each nurse n can work at most $maxHours$ maximum number of hours.

$$\sum_{h=1}^{H} works_{n,h} \leq used_n \times minHours, \forall n \in N \tag{4}$$

Constraint 4: Each nurse n can work at most $maxConsec$ maximum consecutive hours.

$$\sum_{j=i}^{i+maxConsec} works_{n,j} \leq used_n \times maxConsec, \forall n \in N, \forall i \in [1, nHours - maxCosec] \tag{5}$$

Constraint 5: No nurse n can stay at the hospital for more than $maxPresence$ maximum present hours. In other words, if the nurse worked at hour h, he/she cannot work after $h + maxPresence$ hour.

$$WB_{n,h} + WA_{n,h+maxPresence} \leq 1, \forall n \in N, \forall h \in \{h \in H | h \leq nHours - maxPresence\} \tag{6}$$

Constraint 6: No nurse n can rest for more than one consecutive hour.

$\forall n \in N, \forall h \in \{h \in H | h \leq nHours - 1\}$

$$WA_{n,h} \geq WA_{n,h+1} \tag{7}$$

$$WB_{n,h} \leq WB_{n,h+1} \tag{8}$$

$$Rest_{n,h} + Rest_{n,h+1} \leq 1 \tag{9}$$

In order to connect $WB_{n,h}, WA_{n,h}$ and $Rest_{n,h}$ decision variable matrices with a solution matrix $works_{n,h}$, we need to construct following logical equivalence:

$$Rest_{n,h} == (1 - works_{n,h}) - (1 - WA_{n,h}) - (1 - WB_{n,h}) \tag{10}$$

Which means:

If $Rest_{n,h} = 1$, then $(1 - works_{n,h}) - (1 - WA_{n,h}) - (1 - WB_{n,h}) = (1 - 0) - (1 - 1) - (1 - 1) = 1$.

If $Rest_{n,h} = 0$, then $(1 - works_{n,h}) - (1 - WA_{n,h}) - (1 - WB_{n,h}) = (1 - 0) - (1 - 1) - (1 - 0) = 0$

If feasible solution exists, then we get filled matrix $works_{n,h}$ and array of used nurses $used_n$ with minimized value of objective function $\min \sum_{n=1}^{N} used_n$. Which means this optimization problem is solved.

3 Metaheuristics

Integer Linear program guarantees to find an optimal solution if any feasible solution exists. However, this can be quite time consuming for medium and big size problems. That's why it's necessary to introduce some heuristics and solve this combinatorial optimization problem using them. Which assum-

ingly should give a worse solution than OPL, but faster. We will consider two meta-heuristics: **GRASP** and **BRKGA**.

3.1 GRASP

The essence of GRASP is to select a candidate for solution element from a set of candidate elements using some criteria, which will be discussed later. First of all we need to generate that candidate set. The working nurse might be viewed as a boolean array, where *1* means nurse works on that hour, and *0* means nurse doesn't work. An example nurse boolean array might look like that: [0, 1, 1, 1, 1, 0, 0, 0]. And this working nurse is exactly the candidate element for solution. In order to increase time efficiency and solution accuracy, we generate all possible elements for hours $h \in [1, 19]$ and save it locally. Then before starting solving problem, we filter that set of candidate elements so that it corresponds to specific constraints, given by parameters. Even if the set doesn't exist it still takes around 8 secs to generate all $2^{19} - 1 = 524287$ possible elements for hour $h = 19$. And after initial generation, we can load elements from computer. This will improve speed of program. However, in case $h \geq 20$, generating all possible candidate elements and storing them locally is highly time and space consuming. So instead, we temporarily generate randomly $4 \times nNurses$ feasible candidate elements, respecting all given constraints. And then use this set to solve our problem. In case of $nNurses = 1800$, it takes around 8 secs to generate this set of feasible element solutions. Our **GRASP** just like any other is divided into two phases *Constructive Phase* and *Local Search Phase*, with modifications of phases themselves.

3.1.1 Constructive Phase

1: **function** CONSTRUCT(gc(.),α)
2: Initialize candidate set C;
3: $\underline{s} = min\{g(t)|t \in C\}$
4: $\overline{s} = max\{g(t)|t \in C\}$
5: $RCL = \{s \in C|gc(s) \leq \underline{s} + \alpha(\overline{s} - \underline{s})\}$
6: Select candidate element x, at random, from RCL
7: **return** x
8: **end function**

So instead of classical **GRASP** *Constructive Phase*, where candiate solution is constructed, we return one candidate element for overall solution. In most of the cases random candidate element is not the best, so we perform a *Local Search* on returned candidate element x.

3.1.2 Local Search Phase

1: **function** LOCALSEARCH(x, C, α, $nHours$)
2: $maxDistance = \sqrt{nHours} \times (1 - \alpha)$
3: $N(x) = \{t \in C | dist(x, t) \leq maxDistance\}$
4: **for all** $n \in N$ **do**
5: **if** $gc(n) < gc(x)$ **then**
6: x = n
7: **end if**
8: **end for**
9: **return** x
10: **end function**

LOCALSEARCH function will return the best element in a neighborhood of candidate element, where neighborhood is defined by Euclidean distance from that element. The radius of neighborhood is also determined by α parameter. In case $\alpha = 0$, we get the greediest approach, which is the best and slowest at the same time.

The difference between classical **GRASP** and **ours**, is that we don't construct the set of candidate solutions, and then perform *Local Search* on that set. But rather randomly return 1 element from constructed RCL and apply local search on 1 element. This will reduce execution time, while returning same locally optimal candidate element. To conclude **GRASP** let's introduce our *Greedy Cost function*

3.1.3 Greedy Cost Function

1: **function** GREEDYCOST(x, demand)
2: Calculate positive hits: $posHits$
3: Calculate negative hits: $negHits$
4: $demand = demand - x$
5: $hitsDiff = posHits - negHits$
6: $cost = e^{-hitsDiff}$
7: **return** $(demand, cost)$
8: **end function**

Above given *Greedy Cost function* will return updated demand and cost of candidate element. In order to understand this *function* let's consider following example:

- $demand = [1, 2, 3, -2, 4, -5, -6]$
- $x = [1, 0, 0, 1, 1, 0, 0]$

Positive Hits is number of times x decreases from positive values of *demand*, while *Negative Hits* is number of times x decreases from negative values of *demand*. So in above given example, $posHits = 2$ and $negHits = 1$. Since our goal is preferebly to decrease positive values of leftover demand, $hitsDiff$ is used. After that, exponential function is applied to $hitsDiff$, because we

want to exponentially decrease/increase cost in case of wrong/right candidate element and choose the maximum cost. Since we have minimization problem, we invert exponential function, i.e. $\max e^{hitsDiff} = \min e^{-hitsDiff}$.

3.2 BRKGA

4 Comparative Results

4.1 GRASP

4.2 BRKGA