

Course project

Shams Methnani & Rodrigo Arias Mallo

January 10, 2018

1 Problem statement

Our problem to solve is to optimally assign nurses to working hours in a hospital such that demand for each hour is met and the fewest number of nurses are used.

This is an NP-hard problem in combinatorial optimization. It can be formulated as an integer program.

We are given a set of nurses, a specification of the demands for each hour as well as a set of constraints that must be satisfied regarding how long each nurse can work.

1.1 Parameters

We are given a set of available nurses of size N , a list $demand_h$ of size H with the number of nurses needed each hour, as well as the constraints that must be satisfied. The parameters are given as follows:

| | |
|---------------|--------------------------------------------|
| $demand_h$ | Demand for each hour h |
| $maxHours$ | Max. number of hours each nurse can work |
| $minHours$ | Min. number of hours each nurse can work |
| $maxConsec$ | Max. consecutive hours each nurse can work |
| $maxPresence$ | Max. hours present at the hospital |

1.2 Decision variables

For the problem, we define three matrices as decision variables. The last one is auxiliary.

The matrix P has the element $P_{n,h} = 1$ if the nurse n is present at the hospital at hour n , also if is working, $W_{n,h} = 1$, otherwise 0. The matrix T is an auxiliary matrix, with the element $T_{n,h} = 1$ if the nurse n is travelling to the hospital at the hour h , otherwise 0.

| Symbol | Type and size |
|--------|--------------------------------------|
| P | Binary matrix of size $N \times H$. |
| W | Binary matrix of size $N \times H$. |
| T | Binary matrix of size $N \times H$. |

1.3 Constraints

Constraint 1 At least demand_h nurses should be working at the hour h .

$$\forall h \in [1, H], \sum_{n \in N} W_{n,h} \geq \text{demand}_h$$

Constraint 2 Each working nurse should work at least minHours .

$$\forall n \in N, \sum_{h \in [1, H]} W_{n,h} \geq \text{minHours} * \sum_{j \in [1, H]} T_{n,j}$$

Constraint 3 Each nurse should work at most maxHours .

$$\forall n \in N, \sum_{h \in [1, H]} W_{n,h} \leq \text{maxHours}$$

Constraint 4a Each nurse should work at most maxConsec consecutive hours.

$$\forall n \in N, \forall h \in [1, H - \text{maxConsec}],$$

$$\sum_{k \in [0, \text{maxConsec}]} W_{n, h+k} \leq \text{maxConsec}$$

Constraint 4b Wrap around case for maxConsecutive hours.

$$\forall n \in N, \forall h \in [1, \text{maxConsec}],$$

$$\sum_{j \in [1, h]} W_{n,j} + \sum_{j \in [H - \text{maxConsec} + h, H]} W_{n,j} \leq \text{maxConsec}$$

Constraint 5 No nurse can stay at the hospital for more than maxPresence hours.

$$\forall n \in N, \sum_{h \in [1, H]} P_{n,h} \leq \text{maxPresence}$$

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

$$\forall n \in N, \forall h \in [1, H-1],$$

$$W_{n,h} + W_{n,h+1} \geq P_{n,h+1}$$

Constraint 7 Working nurses can travel to hospital at most once.

$$\forall n \in N,$$

$$\sum_{h \in [1, H]} T_{n,h} \leq 1$$

Constraint 8a If a nurse is present at current hour and wasn't at previous hour, then they traveled to hospital at current hour.

$$\forall n \in N, \forall h \in [2, H],$$

$$T_{n,h} \geq 1 - P_{n,h-1} + P_{n,h} - 1$$

Constraint 8b Add case for wrap-around

$$\forall n \in N,$$

$$T_{n,1} \geq 1 - P_{n,H} + P_{n,1} - 1$$

Constraint 9 If nurse travels at current hour, then they were not present at previous hour.

$$\forall n \in N, \forall h \in [2, H],$$

$$T_{n,h} \leq 1 - P_{n,h-1}$$

Constraint 10 If nurse travels at current hour, then they are present at current hour.

$$\forall n \in N, \forall h \in [1, H],$$

$$T_{n,h} \leq P_{n,h}$$

2 GRASP

The GRASP method is a metaheuristic in which each iteration consists of two phases: construction and local search.

First, we need to specify the problem in terms of parts that can be selected. The ground set E contains elements that can be selected in order to build a solution. Let S be a possible solution by selecting parts of E , so $S \in 2^E$. We can build F as the set of solutions that are feasible $F \subseteq 2^E$.

In our problem, we need to select the behavior of N nurses. A possible solution can be defined by the selection of elements that describe what a nurse n is doing at the time h . A tuple $t = (n, h, s)$ can code that the nurse n should be at state s in the time h . The state can be 0 if the nurse is not at the hospital, 1 if is working, or 2 if is resting. A feasible solution S should consist of a selection of tuples t such that, they define the behavior of every nurse at every time, and also satisfy the previous constraints.

Furthermore, in order to build a solution, a notion of cost is needed to select among all the possible tuples that can be selected. Lets ignore the constraints in the cost function, and check the feasibility at the end. Iteratively we can select a naive solution by a simple procedure.

We begin with an empty solution, $S = \{\}$. Then we compute the cost of all the possible elements that can be added to the solution. A good candidate will be try to meet the demand by adding nurses that are in the working state.

We create the restricted candidate list (RCL) by selecting working nurses with high probability. Then, one of the possible combinations will be chosen, optimistically assigning one nurse to work in that time. Then we continue adding nurses in a working state. Once the demand is met, then we now will prefer adding to the solution tuples that contain nurses in a non-working state. The probability of such state should now be added with high probability.

The selection of the cost function is the core of the GRASP algorithm, so we need to take some care to guarantee a good behavior.

3 BRKGA

Genetic algorithms borrow mechanisms from nature, namely the concept of survival of the fittest, in order to search a large solution space. A population of solutions is evolved over several generations by iteratively breeding new solutions by mating or *crossing* individuals from previous generations. Biased Randomized-Key Genetic Algorithm (BRKGA) is such an algorithm in which certain members of the population are selected for crossover with higher

probability.

We represent our problem with a set of individuals with an associated *chromosome*, each of which encodes a solution.

Each individual or chromosome has associated with it a fitness score, which will determine if they are an *elite individual* or *non-elite individual*. We can now add a bias to the selection of the parentage for next generation individuals the selecting a parent from the elite set with a higher probability in order to pass on desirable characteristics.

Each chromosome is represented as a string of genes. Each gene takes on a particular value from an alphabet.

Our matrix encodes start-hour, present-hour, number-breaks, break-score 3 scalars and an array of scores in order to place breaks.

3.1 Chromosome structure

The solution is codified by using three scalars p , s and b and a vector v of size H for each nurse. The scalars codify the number of hours p that the nurse is present at the hospital, the first hour s at which is at the hospital, and the number of breaks b while is present.

The vector v is used from 1 to the number of present hours p . Each element stores a score that will be used to determine the preferred hours in which a break should be set. Lower scores will be selected first, filling the breaks up to b .

The decision whether a nurse works or not, is based on p , if the presence hours are less than *minHours* the nurse doesn't start working.

The chromosome structure is then matrix of N rows and $3 + H$ columns, but flattened into a long vector of size $N \times 3 + H$.

3.2 Decoder

The decoder part of the algorithm, which is the only part that depends on the problem, computes the fitness of each individual in the population. The fitness $f(p, s, b, v)$ is a heuristic value that estimates the objective, and has to be minimized. Is computed by adding five badness values BN , BD , BH , BC and BF together.

In order to compute the values, we first need to determine when each nurse works. By using p and s we have the interval that they are present. And using the indices of the first b lower values of v , we now where they rest.

The number of nurses needed is the first value BN , which takes into account that we want to minimize the number of nurses used.

The value BD measures the demand left that is not met. Is computed at each hour h as the difference of the problem demand and the nurses offer (the working hours provided by the working nurses) as $BD = 100 \sum_h \text{demand}_h - \text{offer}_h$.

The value $BH(n)$ measures the error of each nurse n , when they work more than maxHours or when they work less than minHours , we use $w = p - b$ to compute the number of working hours. If they work more hours than allowed, $BH(n)$ is set to the number of exceeding hours $w - \text{maxHours}$. If they work less than allowed, $BH(n)$ is set to the number of hours left $\text{minHours} - w$. Finally, we get $BH = \sum_n 200 + 20BH(n)$.

Note that we use the floating point representation of the values p , s , b and v when they are part of the fitness function. So we can allow that lower differences of each gene produce a change in the fitness function. Otherwise, we use the integer representation of the parameters.

The value BC test if there is any problem with the maximum number of consecutive hours that each nurse is working. For each nurse n the maximum number of consecutive hours M is computed, and if is greater than maxConsec , $BC(n)$ is set to $100(M - \text{maxConsec}) - 10b$. We use the breaks to prefer those solutions that include more breaks. Finally we have $BC = \sum_n BC(n)$.

The value BF measures the distance of the non-working nurses to the hours that have some demand left. The distance is computed for each hour with demand left h to the closest hour c that the nurse would start working if p was at least minHours . Is the start hour s when $s > h$ and the end hour $s + p$ when $s + p < h$. So, if the nurse don't cover the hour h , $BF(n) = h - c$. However, if the nurse could cover the hour, without changing start, we only need to take into account the distance to minHours , so $BF(n) = \text{minHours} - p$. The final value is the sum for all nurses that are non-working, $BF = \sum_n BF(n)$.

The final fitness value is just the sum of all the intermediate values:

$$f(p, s, b, v) = BN + BD + BH + BC + BF$$

The fitness is stored in an array, computed for each individual of the population, and returned to the BRKGA algorithm, in order to continue the iteration.

4 Results

The three solvers were compared solving equal problems, with a big number of nurses.

| Seed | Nurses | ILP | | GRASP | | BRKGA | |
|------|--------|------|------|-------|-------|-------|---------|
| | | Obj. | Time | Obj. | Time | Obj. | Time |
| 8 | 90 | 41 | 61.1 | 52 | 62.9 | ? | 83.1809 |
| 9 | 90 | 39 | 61.2 | ? | 619.3 | 42 | 71.2341 |
| 10 | 90 | 39 | 61.2 | 56 | 62.2 | 42 | 51.6024 |
| 11 | 90 | 31 | 61.2 | 43 | 63.4 | 44 | 46.2278 |
| 12 | 90 | 39 | 61.2 | 46 | 127.4 | 41 | 96.5402 |
| 13 | 90 | 36 | 61.1 | ? | 638.2 | ? | 81.3786 |
| 14 | 90 | 39 | 61.1 | 51 | 95.3 | ? | 71.1666 |
| 15 | 90 | 31 | 61.2 | 42 | 32.2 | 43 | 65.3363 |
| 16 | 90 | 35 | 61.1 | 56 | 189.3 | 44 | 45.5936 |
| 17 | 90 | 35 | 61.2 | 57 | 31.5 | 42 | 63.4561 |

Table 1: Results with the different methods

5 Structure of the project

5.1 The OPL model

The ILP problem was first written in OPL by using a small example, and it can be found in the `opl/` folder. The `proj.mod` file contains the model of the problem, and the example data is stored in `proj.dat`. A small python script and a CPLEX batch file help the extraction of the solution. The Makefile is designed to use `oplrun` and `cplex` to solve the problem and present the solution. The `oplrun` program evaluates the model and builds the file `sol.lp`, which is solved by the `cplex` program. The script `cplex-save.txt` provide CPLEX the instructions to save the solution in `sol.xml` in order to be read by the python script `cplex-save.txt`:

```
% cd opl/
% make
...
cplex -f cplex-save.txt

Welcome to IBM(R) ILOG(R) CPLEX(R) Interactive Optimizer 12.6.0.0
...
Root relaxation solution time = 0.00 sec. (0.15 ticks)
...
python read-sol.py
NStart
[[0 0 0 0 1 0]
 [1 0 0 0 0 0]
```

```

[1 0 0 0 0 0]]
NWorking
[[0 0 0 0 1 1]
 [1 1 0 0 0 0]
 [1 1 0 0 0 0]]
NPresent
[[0 0 0 0 1 1]
 [1 1 0 0 0 0]
 [1 1 0 0 0 0]]

```

As we need to run and evaluate a lot of problems, this method is a bit complex, as requires the problem to be defined in the OPL data language. We opted for the use of the PuLP python package to communicate with CPLEX, in order to write the problem and read the solution as well as the objective value, more easily.

5.2 The PuLP model

The PuLP package provides a almost transparent interface to work with multiple solvers. We used the CPLEX solver by default, as it seems to be the fastest.

The same constraints in `opl/proj.mod` were translated into python. The file `src/lp.py` contains the ILP model in python. A class `Solution` provides a generic method `solve` to be implemented, as well as an instance to a problem to be solved.

The instance generator is built in the `Problem` class, located in `nurses.py`

5.3 Instance generator

Random instances of the problem are generated by using the parameters: `seed` the random seed, `N` the number of nurses and `H` the hours in a day, set by default to 24.

We used a random distribution, with more expected value in the daily hours, behaving similarly to a real case. Once a instance problem generated can be solved by any solver. An example problem:

```

% python nurses.py
Problem(seed=1, N=10, H=24)
-----
hour  0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 2 2 2 2
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3
-----

```



```
dem.  1 1 0 0 1 0 0 2 1 2 2 2 3 2 3 1 1 0 1 0 0 0 1 0
```

```
maxConsec    : 7  
maxPresence  : 9  
maxHours     : 8  
minHours     : 8
```

6 How to run

The ILP, GRASP and BRKGA solvers are coded in a extension of the `Solution` class, placed in `src/`, in the files `lp.py`, `grasp.py` and `brkga.py`, respectively.

Each solver is iteratively choosen to solve the same problem. Once all the solutions are computed, a new random problem is generated and solved again. The file `main.py` produces the table 1. Can be executed as:

```
% python main.py
```