Today, I would like to introduce about our mini-project, which is

Nurse Rostering Problem

Fundamentals of Optimization Mini-project

Supervisor: **Professor Phạm Quang Dũng**

Our team includes three members:

1. Tạ Ngọc Minh - 20214918
2. Lê Ngọc Bình - 20214878
3. Ngô Việt Anh - 20214875

Table of contents

Table of contents

1 Introduction

Rostering problem, or Nurse Scheduling Problem is considered as a NP-hard optimization problem, which has the question is:

- There are *N* nurses need to be assigned to work shifts for *D* days. Each day is divided into 4 shifts: morning, noon, afternoon, and night.

1. A nurse can only work most 1 shift per day.
2. If a nurse work night shift the day before, she can rest the next day.
3. Each shift in each day has at least *a* nurses and at most *b* nurses.
4. *F(i)*: list of days off for nurse *i*.

The problem is: **Minimize the maximum number of night shifts assigned to a certain nurse.**

Table of contents

2

Modeling

Input:

- Number of nurses: N
- Number of days: D
- Number of shifts: S = 4
- Min. nurses work in a shift: a
- Max. nurses work in a shift: b
- List of day off for nurses: dayoff[n][d]

Modeling

Case 1: Binary string with length N*D*S

<u>Example:</u> Day 1 of 4 nurses.

assign =

1	0	0	0	0	1	0	0	0	1	0	0	0	0	0	1
M	A	E	N	M	A	E	N	M	A	E	N	M	A	E	N

We denote:

M = morning, A = afternoon, E = evening, N = night

Modeling

Case 1: Binary string with length `N*D*S`

<u>**Objective function:**</u> `Min(max(sum(assign[n,d,4])))`

<u>Conditions:</u> Here, we only consider `assign[n,d,s]` is representative of the nurse `n` work in day d, shift `s`.

```
1. sum(assign[n,d,s]) <= 1,            s < 4.
2. If assign[n,d,4] = 1 then assign[n,d,s] = 0,    s < 4.
3. a <= sum(assign[n,d,s]) <= b,    n < N, s < 4.
4. assign[n,d,s] = 0,                  s < 4, and d in dayoff[n].
```

Modeling

Case 2: A string of 5 variables [0,4] with length N*D

<u>Example:</u> 2 days of 8 nurses.

```
assign =
```

1	2	3	0	0	4	0	1	2	0	4	0	1	1	3	3
1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8

Modeling

Case 2: A string of 5 variables [0,4] with length N*D

Objective: Min(max(sum(assign[n,d]) if assign[n,d] = 4)

Conditions: Here, we only consider assign[n,d] = s is representative of the nurse n work in day d, shift s.

1. The first condition is already satisfied.
2. If assign[n,d] = 4 then assign[n,d+1] = 0.
3. a <= sum(assign[n,d]) <= b, n < N.
4. assign[n,d] = 0, n < N and d in dayoff[n].

Table of contents

Solution

Techniques used:

- Backtracking,
- Constraint Programming,
- Linear Programming,
- Heuristics (Greedy algorithm),
- Local Search (Hill-climbing algorithm), and
- Meta-heuristics (Genetic Algorithm).

3.1 Backtracking algorithm

Here, we use case 1 of Modeling for the sake of time.

```
Initialize arr, length N*D, filled with 0.

def generateAllSolutions(n, arr, i, N, D, alpha, beta):
    if i == n:
        check if the current arr configuration violates any constraints
        if not, count the number of night shifts
        update best value (if applicable)

    else:
        continue generating new solutions

    for j in [0:4] do
        if arr[i] != -1:
            arr[i] = j
            generateAllSolutions(n, arr, i+1, N, D, alpha, beta)

        if arr[i] == -1 and i+2 <= n:
            generateAllSolutions(n, arr, i+2, N, D, alpha, beta)
```

Backtracking algorithm

Here, we use case 1 of Modeling for the sake of time.

```
def check_night_shift_condition:
    if worked the night shift and go to work the next day:
        return False
    else:
        return True

def limit_nurses:
    if number of nurses in shift in range(a,b):
        return True
    else:
        return False
```

3.2 Linear & Constraint Programming

Since LP and CP has the same way of implementation using case 1 with OR-tools, we will present one for both.

Creating conditions:

1.4.1. A nurse can be assigned to only one shift per day.

```
for n in range(N):
    for d in range(D):
        p = 0
        for s in range(4):
            p += assign[n][d][s]
        model.Add(p <= 1)
```

1.4.2. Each shift has min a nurses and max b nurses.

```
for d in range(D):
    for s in range(4):
        p = 0
        for n in range(N):
            p += assign[n][d][s]
        model.Add(p >= a)
        model.Add(p <= b)
```

1.4.3. A nurse worked night shift in the previous day has the next day off.

```
for n in range(N):
    for d in range(1,D):
        p = 0
        for s in range(4):
            p += assign[n][d][s]
        model.Add(p + assign[n][d-1][3] >= 1-dayoff[n][d])

d = 0
for n in range(N):
    p = 0
    for s in range(4):
        p += assign[n][d][s]
    model.Add(p >= 1 - dayoff[n][d])
```

3.4 Heuristics – Greedy algorithm

Here, we use case 2 of Modeling.

```
def schedule_employees:
    Initialize schedule
    For each day:
        Generate new availableNurse list
        number_nurses_nightshift = a
        number_nurses_othershifts = min((s - count_night) // 3, b)
        sort the nurses in night shifts ASC
        assign nurses to work on shifts of the day with above numbers
        for i in range(1, 5):
            if i == 4:
                assigned_nurses = avaiableNurse[:number_nurses_nightshift]
                for nurse in assigned_nurses:
                    schedule[nurse][day] = i
            else:
                assigned_nurses = avaiableNurse[number_nurses_nightshift
                                                 + number_nurses_othershifts * (3 - i)
                                                : number_nurses_nightshift
                                                 + number_nurses_othershifts * (4 - i)]
                for nurse in assigned_nurses:
                    schedule[nurse][day] = i
        remain_part = avaiableNurse[number_nurses_nightshift + number_nurses_othershifts * 3:]
        if len(remain_part) > 0:
            for k in range(len(remain_part)):
                schedule[remain_part[k]][day] = k + 1
    return schedule
```

3.5 Local search – Hill-climbing

Here, we use case 2 of Modeling.

We implement two versions of hill-climbing:

1. Use PyCBLS library.
2. Do not use PyCBLS library.

```
def evaluate(schedule):
    return max(night_shift_counts)

def generate_neighbors(schedule):
    Generate all possible neighbors by swapping the shift of 2 nurses on 1 day

def hill_climbing:
    Generate an initial solution
    while True:
        generate_neighbors(current_state, D)
        Evaluate the neighbors and find the best one
        If the best neighbor > current state, move;
        else: return the current state
```

3.6 Meta-heuristic – Genetic algorithm

Here, we use case 1 of Modeling and deap library.

```
Determine objective function = max night shifts

Assign number of generation to 0 (t = 0)
Randomly create individuals in initial population P(t)
Evaluate individuals in population P(t) using objective function
While termination criterion is not satisfied do:
    t += 1
    select the individuals to population P(t) from P(t-1)
    change individuals of P(t) using crossover and mutation
    evaluate individuals in population P(t) using objective function

Return the best individual
```

Genetic parameters:

```
POPULATION_SIZE = 200  # 300
P_CROSSOVER = 0.95  # probability for crossover
P_MUTATION = 0.05  # probability for mutating an individual
MAX_GENERATIONS = 1000  # 200
HALL_OF_FAME_SIZE = 30
HARD_CONSTRAINT_PENALTY = 10
```

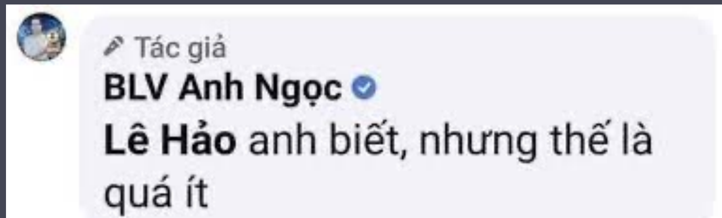
Table of contents

4

Result

This is our experimental result after 9 days running model on
Google Cloud VM with 8 vCPU and 30GB RAM

Nobody asks but I want to flex that it costs $300 per month.
(but I had coupons, so I rented it for free



4 Result

Set	Case	Des.	Running time and optimal solution (average for LS, GR and GA)						
			BA	LP	CP	LS1	LS2	GR	GA
1	0	N = 15, D = 10, a = 1, b= 6	230.523	0.087166	0.108412	0.588605	0.182111	0.001101	41.3822
			1	1	1	1	1	1	1
	1	N = 30, D = 20, a = 3, b = 10	454.245	0.141002	0.125880	22.3629	0.188720	0.001905	143.753
			1	1	1	1	1	1	1
	2	N = 30, D = 50, a = 1, b = 12	1135.94	0.146252	0.158660	29.4207	0.248573	0.000794	192.152
			3	3	3	3	3	3	3
	3	N = 50, D = 50, a = 8, b = 25	1912.42	0.174401	0.234587	51.6095	0.413216	0.000925	329.541
			3	3	3	4	4	3	4
	4	N = 70, D = 90, a = 5, b = 25	4820.21	0.227001	0.443464	63.6780	1.31960	0.001008	451.549
			3	3	3	5	4	3	5
	5	N = 60, D = 90, a = 5, b = 25	4467.43	0.380919	1.372727	113.635	1.77443	0.001296	1620.32
			-	6	6	11	9	6	10
	6	N = 80, D = 100, a = 10, b = 35	∞	0.669191	3.220695	140.897	10.7094	0.002370	2398.08
			-	8	8	10	10	8	9
	7	N = 70, D = 100, a = 10, b = 40	∞	0.716404	4.934985	621.624	12.4616	0.002027	4031.19
			-	5	5	7	7	5	7
	8	N = 150, D = 200, a = 20, b = 55	∞	1.07189	6.375451	724.795	18.1443	0.002389	12503.4
			-	9	9	12	13	9	10
	9	N = 300, D = 500, a = 50, b = 100	∞	1.274229	6.614369	989.702	76.3162	0.002904	25395.6
			-	10	10	14	15	10	13

4 Result

Set	Case	Des.	Running time and optimal solution (average for LS, GR and GA)						
			BA	LP	CP	LS1	LS2	GR	GA
2	0	N = 500, D = 1000, a = 60, b =350	∞	0.145413	0.541119	54.1104	0.167574	0.039692	330.232
			-	2	2	2	2	2	2
	1	N = 500, D = 1200, a = 70, b =300	∞	0.209149	0.472759	102.492	0.336166	0.055338	817.241
			-	3	3	5	5	3	3
	2	N = 700, D = 1400, a = 90, b = 400	∞	0.618743	4.793574	473.850	7.91741	0.064136	3484.44
			-	6	6	9	9	6	10
	3	N = 900, D = 1800, a = 100, b = 500	∞	0.813298	4.952744	565.537	15.0892	0.063667	3571.30
			-	8	8	9	10	8	10
	4	N = 1000, D = 2000, a = 120, b = 800	∞	3.406689	40.111719	1673.32	22.6402	0.054685	6080.05
			-	13	13	15	16	13	15
	5	N = 1200, D = 2000, a = 120, b = 800	∞	5.143919	8.737315	2095.31	28.5830	0.046169	6957.19
			-	12	12	17	17	12	15
	6	N = 1500, D = 2500, a = 200, b = 1000	∞	6.026293	55.136954	2401.49	39.0578	0.061820	8205.12
			-	17	17	22	24	17	19
	7	N = 2000, D = 3000, a = 300, b = 1200	∞	327.985	3558.94	9618.07	50.3028	0.072856	15702.8
			-	50	50	56	58	50	55
	8	N = 2000, D = 4000, a = 300, b = 1200	∞	3465.81	36294.6	22641.5	72.8448	0.142938	26309.6
			-	67	67	73	75	67	75
	9	N = 3000, D = 5000, a = 500, b = 2500	∞	38551.1	60021.2	47268.2	114.184	0.420681	53607.1
			-	140	140	146	149	140	154

Table of contents

5 Conclusion

- The problem can be solved totally or partially using these above techniques.
- The performance of greedy algorithm is proved to has the fastest speed and accuracy.
- For small test case, CP and LP is proved to has better solution in the smaller time.
- For big test case, genetic algorithm has better solution quality (low variations) in the accepted time, however, the low speed of GA is caused by the **deap** library.
- Using local search without **PyCBLS** can have faster solution, but the solution quality is worse.

Thank you for joining us today!

Nurse Rostering Problem

Fundamentals of Optimization Mini-project

Supervisor: **Professor Phạm Quang Dũng**

If you have any question, please ask me (ChatGPT) instead of our team member <3

1. Tạ Ngọc Minh - 20214918
2. Lê Ngọc Bình - 20214878
3. Ngô Việt Anh - 20214875