Nurse Rostering Problem

January 14, 2019

1 Norse Rostering Problem

1.1 Import library

```
In [1]: from __future__ import division
        from __future__ import print_function
        import random
        import copy
        import timeit
```

1.2 Data

```
In [2]: ability_nurses = [[0, 1, 2], [3, 2, 1], [1, 3, 2], [1, 2, 3], [2, 3, 1], [3, 3, 3]]
        num_shifts = 4
        num_nurses = 6
        num_days = 3
        n_0 = 2
        n_1 = 2
        n_2 = 1
        w_1 = 2
        w_2 = 3
        w_3 = 3
        w_4 = 3
```

1.3 Initiation global variables

```
In [3]: max_num_night_shift = 1
        max_threshold_tabu_search = 10
        num_back_track = 0
        max_back_track = 50000
        all_nurses = range(num_nurses)
        all_shifts = range(num_shifts)
        all_days = range(num_days)
        domain = {}
        decision_variable = {}
        primary_solution = []
```

1.3.1 Initiation domain and decision varibales

```
In [4]: def initiation_domain_variable():
            for n in all_nurses:
                for d in all_days:
                    domain[(n, d)] = set(all_shifts)
                    decision_variable[(n, d)] = -1
```

1.4 Define methods which are used in project

Print solution

```
In [5]: def print_solution(source_to_print):
            for n in all_nurses:
                for d in all_days:
                    print("%i" % source_to_print[(n, d)], end=" ")
                print()
            print("Total ability = %i" % get_value_of_function_objective(source_to_print, abil
```

Get number of night shift of an nurse "n" from day 1 -> d

```
In [6]: def get_number_night_shift_of_nurse(n, d, source_to_check):
            sum_night_shift = 0
            for i in range(d):
                sum_night_shift = sum_night_shift + (1 if source_to_check[(n, i)] == 2 else 0)

            return sum_night_shift
```

Calculating value of an solution. This is denoted for function objective F

```
In [7]: def get_value_of_function_objective(source_to_get_value, data):
            sum_ability = 0
            for n in all_nurses:
                for d in all_days:
                    working_shift = source_to_get_value[(n, d)]
                    if working_shift != 3:
                        sum_ability = sum_ability + data[n][working_shift]

                    # penalty for night -> morning:
                    if working_shift == 0 and (d > 0 and source_to_get_value[(n, d - 1)] == 2)
                        sum_ability = sum_ability - w_1

                    # bonus for free free
                    if working_shift == 3 and (d > 0 and source_to_get_value[(n, d - 1)] == 3)
                        sum_ability = sum_ability + w_2

                    # morning free
                    if working_shift == 3 and (d > 0 and source_to_get_value[(n, d - 1)] == 0)
                        sum_ability = sum_ability + w_3
                    # free night
                    if working_shift == 2 and (d > 0 and source_to_get_value[(n, d - 1)] == 3)
                        sum_ability = sum_ability + w_4

            return sum_ability
```

Checking constraint minimum nurse required for one day

```
In [8]: def check_minimum_nurses_required(source_to_check):
            for d in all_days:
                nurse_morning_shift = 0
                nurse_afternoon_shift = 0
                nurse_night_shift = 0

                for n in all_nurses:
                    shift_type = source_to_check[(n, d)]
                    # sum morning shift
                    if shift_type == 0:
                        nurse_morning_shift += 1
                    # sum afternoon shift
                    elif shift_type == 1:
                        nurse_afternoon_shift += 1
                    # sum night shift
                    elif shift_type == 2:
                        nurse_night_shift += 1

                if nurse_morning_shift < n_0 or nurse_afternoon_shift < n_1 or nurse_night_shi
                    return False

            return True
```

Check constraint number night shift in a period should be less than "t"

```
In [9]: def check_number_night_shift(source_to_check):
            for n in all_nurses:
                if get_number_night_shift_of_nurse(n, num_days, source_to_check) > max_num_nig
                    return False

            return True
```

Check constraint ability to perform a task of nurse "n" in shift "s"

```
In [10]: def check_ability_perform_shift(n, s, data):
             if s == 3:
                 return True
             else:
                 return data[n][s] != 0
```

Non binary checking method for nurse "n" in day "d"

```
In [11]: def non_binary_checking_cp(n, d):
             domain_copy = copy.deepcopy(domain)
             current_number_night_shift = get_number_night_shift_of_nurse(n, d, domain)
             for i in range(d + 1, num_days):
                 for s in domain[(n, i)]:
                     if current_number_night_shift + (1 if s == 2 else 0) > max_num_night_shif
                         domain_copy[(n, i)].remove(s)

                     if not domain_copy[(n, i)]:
                         return False

             return True
```

Forward checking method

```
In [12]: def check_forward(n, d):
             global num_back_track

             value_random = random.sample(domain[(n, d)], 1)[0]

             if not check_ability_perform_shift(n, value_random, ability_nurses):
                 num_back_track = num_back_track + 1
                 return 2

             if get_number_night_shift_of_nurse(n, d, decision_variable) + (1 if value_random
                 num_back_track = num_back_track + 1
                 return 2

             decision_variable[(n, d)] = value_random

             if n == num_nurses - 1 and d == num_days - 1:
                 if check_minimum_nurses_required(decision_variable):
                     primary_solution.append(copy.deepcopy(decision_variable))
                     return 1

                 num_back_track = num_back_track + 1
                 return 2

             next_day = (d + 1) % num_days
             next_nurse = n + (1 if next_day % num_days == 0 else 0)
             decision = check_forward(next_nurse, next_day)
             if decision != 0:
                 return decision

             decision_variable[(n, d)] = -1
             num_back_track = num_back_track + 1
             if num_back_track >= max_back_track:
                 return 2

             return 0
```

Tabu search local adjustment

```
In [13]: def tabu_seach_local_adjustment(source_to_adjust):
             result = source_to_adjust.copy()
             best = get_value_of_function_objective(source_to_adjust, ability_nurses)

             for d in all_days:
                 for i in all_nurses:
                     for k in all_nurses:
                         if i != k:
                             current_decision_variable = copy.deepcopy(result)
                             current_decision_variable[(i, d)], current_decision_variable[(k, d
                             if check_minimum_nurses_required(current_decision_variable):
                                 if check_number_night_shift(current_decision_variable):
                                     if check_ability_perform_shift(i, current_decision_variabl
                                         if check_ability_perform_shift(k, current_decision_va
                                             current_value = get_value_of_function_objective(cu
                                             if current_value > best:
                                                 best = current_value
                                                 result = current_decision_variable.copy()

             return result
```

Tabu search main

```
In [14]: def tabu_search(source_to_search):
             result = copy.deepcopy(source_to_search)
             best = get_value_of_function_objective(result, ability_nurses)
             threshold = 0
             while threshold < max_threshold_tabu_search:
                 tmp_solution = tabu_seach_local_adjustment(source_to_search)
                 tmp_value = get_value_of_function_objective(tmp_solution, ability_nurses)
                 if tmp_value > best:
                     best = tmp_value
                     result = copy.deepcopy(tmp_solution)
                 source_to_search = tmp_solution.copy()

                 threshold = threshold + 1

             return result
```

Main method

```
In [15]: def main():
             # declaration
             global domain

             # initiation
             initiation_domain_variable()

             # FC_CBJ_NONBINARY_CP algorithm - first phase
             domain_backup = domain.copy()
             while num_back_track < max_back_track:
                 decision = check_forward(0, 0)
                 if decision == 2:
                     domain = copy.deepcopy(domain_backup)

             # Enhance result by tabu-search - second phase
             if primary_solution:
                 result = tabu_search(primary_solution[0])
                 best = get_value_of_function_objective(result, ability_nurses)
                 for i in range(1, len(primary_solution)):
                     primary_solution[i] = tabu_search(primary_solution[i])
                     current = get_value_of_function_objective(primary_solution[i], ability_nu
                     if best < current:
                         best = current
                         result = primary_solution[i].copy()

                 print_solution(result)
             else:
                 print("No solution found")


         if __name__ == '__main__':
             # Start timer
             start = timeit.default_timer()

             # Start solving problem
             main()

             # Stop timer
             stop = timeit.default_timer()

             # Print time execution
             print('Time: ', stop - start)
```

```
2 1 1
0 0 0
1 1 1
3 2 1
0 3 2
```

```
1 0 0
Total ability = 48
Time:  12.244135400456772
```