In [405]:

```
import random
from pprint import pprint
import math
from copy import deepcopy

import numpy as np

from utils import read_data
from constants import FILENAME_EXAMPLE_1, FILENAME_EXAMPLE_4
```

In [2]:

```
input_data = read_data(FILENAME_EXAMPLE_4)
#input_data = read_data(FILENAME_EXAMPLE_4)

days    = input_data['length_of_schedule']
ne      = input_data['number_of_employees']
ns      = input_data['number_of_shifts']
demand  = input_data['temporal_requirements_matrix']
sn      = input_data['shift_name']
ss      = input_data['start_shift']
ls      = input_data['length_shift']
min_ls  = input_data['min_length_of_blocks']
max_ls  = input_data['max_length_of_blocks']
min_do  = input_data['min_days_off']
max_do  = input_data['max_days_off']
min_lw  = input_data['min_length_work_blocks']
max_lw  = input_data['max_length_work_blocks']
nf2     = input_data['nr_sequences_of_length_2']
nf3     = input_data['nr_sequences_of_length_3']
f2      = input_data['not_allowed_shift_sequences_2']
f3      = input_data['not_allowed_shift_sequences_3']
```

In [3]:

```
shifts = ns + 1
day, afternoon, night, dayoff = 1, 2, 3, 4
code = sn + ['-']
```

In [47]:

```
def generate_random_solution(input_data):
    return [[random.choice(code) for d in range(days)] for e in range(ne)]
```

In [5]:

```
def demand_constraint(result):
    for e in range(ne):
        for d in range(days):
            sum_day, sum_afternoon, sum_night = 0, 0, 0
            if result[e][d] == 'D': sum_day += 1
            elif result[e][d] == 'A': sum_afternoon += 1
            elif result[e][d] == 'N': sum_night += 1
        if sum_day >= demand[0][d] and sum_afternoon >= demand[1][d] and sum_night
            pass
        else:
            return False
    return True
```

In [406]:

```
def update_demand_constraint(result):
    updated_result = deepcopy(result)
    for d in range(days):
        sum_day, sum_afternoon, sum_night = 0, 0, 0
        for e in range(ne):
            if updated_result[e][d] == 'D': sum_day += 1
            elif updated_result[e][d] == 'A': sum_afternoon += 1
            elif updated_result[e][d] == 'N': sum_night += 1

        if sum_day >= demand[0][d]: pass
        else: updated_result[random.randint(0, ne-1)][d] = 'D'
        if sum_afternoon >= demand[1][d]: pass
        else: updated_result[random.randint(0, ne-1)][d] = 'A'
        if sum_night >= demand[2][d]: pass
        else: updated_result[random.randint(0, ne-1)][d] = 'N'
    return updated_result
```

In [7]:

```
#day off constraint
def day_off_constraint(result):
    for e in range(ne):
        count_dayoff = 0
        for d in range(days):
            if result[e][d] == '-': count_dayoff += 1
        if min_do <= count_dayoff <= max_do: pass
        else: return False
    return True
```

n [407]:

```
f constraint
ate_day_off_constraint(result):
ated_result = deepcopy(result)
 e in range(ne):
 count_dayoff = 0
 for d in range(days):
     if updated_result[e][d] == '-': count_dayoff += 1
 if count_dayoff >= min_do: pass
 else: updated_result[e][random.randint(0, days-1)] = '-'
 if count_dayoff <= max_do: pass
 else: updated_result[e][random.randint(0, days-1)] = random.choice(list(set(code)
urn updated_result
```

In [9]:

```
#working days in a row constraint
def length_work_blocks_constraint(result):
    for e in range(ne):
        count_consecutive = 0
        min_flag, max_flag = False, False
        for d in range(days - 1):
            if result[e][d] != '-' and result[e][d+1] != '-':
                count_consecutive += 1
                if count_consecutive >= min_lw - 1:
                    min_flag = True
                    if count_consecutive <= max_lw - 1:
                        max_flag = True
            else:  count_consecutive = 0
        if min_flag and max_flag: pass
        else: return False
    return True
```

In [408]:

```
#working days in a row constraint
def update_length_work_blocks_constraint(result):
    updated_result = deepcopy(result)
    for e in range(ne):
        count_consecutive = 0
        min_flag, max_flag = False, False
        for d in range(days - 1):
            if updated_result[e][d] != '-' and updated_result[e][d+1] != '-':
                count_consecutive += 1
                if count_consecutive >= min_lw - 1:
                    min_flag = True
                    if count_consecutive <= max_lw - 1:
                        max_flag = True
            else:  count_consecutive = 0

        if min_flag: pass
        else:
            count_concecutive = 0
            for d in range(days):
                if updated_result[e][d] == '-' and count_concecutive <= max_lw - 1
                    updated_result[e][d] = random.choice(list(set(code) - set('-')
                count_concecutive += 1
        if max_flag: pass
        else: updated_result[e][random.randint(0, days-1)] = '-'

    return updated_result
```

In [11]:

```
#forbidden shifts constraint
def forbidden_constraint2(result):
    if f2 == []: return True
    if f2 != []:
        for e in range(ne):
            for d in range(days - 1):
                for f in f2:
                    if result[e][d] == f[0] and result[e][d+1] == f[1]: return Fal
        return True
```

In [409]:

```
#forbidden shifts constraint
def update_forbidden_constraint2(result):
    updated_result = deepcopy(result)
    if f2 == []: return updated_result
    if f2 != []:
        for e in range(ne):
            for d in range(days - 1):
                for f in f2:
                    if updated_result[e][d] == f[0] and updated_result[e][d+1] ==
                        for c in code:
                            if [f[0], c] not in f2:
                                updated_result[e][d+1] = c
                                break
        return updated_result
```

In [410]:

```
#forbidden shifts constraint
def update_forbidden_constraint2_2(result):
    updated_result = deepcopy(result)
    if f2 == []: return updated_result
    if f2 != []:
        for e in range(ne):
            for d in range(days - 1):
                for f in f2:
                    if updated_result[e][d] == f[0] and updated_result[e][d+1] ==
                        for c in code:
                            if [c, f[1]] not in f2:
                                updated_result[e][d] = c
                                break
        return updated_result
```

In [15]:

```
def forbidden_constraint3(result):
    if f3 == []: return True
    if f3 != []:
        for e in range(ne):
            for d in range(days - 2):
                for f in f3:
                    if result[e][d] == f[0] and result[e][d+1] == f[1] \
                                            and result[e][d+2] == f[2]: return Fal
        return True
```

In [411]:

```
def update_forbidden_constraint3(result):
    updated_result = deepcopy(result)
    if f3 == []: return updated_result
    if f3 != []:
        for e in range(ne):
            for d in range(days - 2):
                for f in f3:
                    if result[e][d] == f[0] and result[e][d+1] == f[1] \
                                            and result[e][d+2] == f[2]:
                        for c in code:
                            if [c, f[1], f[2]] not in f3:
                                updated_result[e][d] = c
                                break

        return updated_result
```

In [412]:

```
def update_forbidden_constraint3_2(result):
    updated_result = deepcopy(result)
    if f3 == []: return updated_result
    if f3 != []:
        for e in range(ne):
            for d in range(days - 2):
                for f in f3:
                    if result[e][d] == f[0] and result[e][d+1] == f[1] \
                                            and result[e][d+2] == f[2]:
                        for c in code:
                            if [f[0], c, f[2]] not in f3:
                                updated_result[e][d] = c
                                break

        return updated_result
```

In [61]:

```
def eval_solution(solution):
    score = 0
    c1, c2, c3, c4, c5 = demand_constraint, \
                         day_off_constraint, \
                         length_work_blocks_constraint, \
                         forbidden_constraint2, \
                         forbidden_constraint3
    if c1(solution): score += 50
    if c2(solution): score += 15
    if c3(solution): score += 15
    if c4(solution): score += 10
    if c5(solution): score += 10

    return score
```

In [44]:

```
def exp_probability(solution, neighbor, T):
    return math.exp((eval_solution(neighbor) - eval_solution(solution))/T)
```

In [452]:

```
def simulated_annealing(input_data, t_max, T_max, r, termination_condition):
    t = 0
    tc = 0
    score = 100
    T = T_max

    solution = generate_random_solution(input_data)

    while t < t_max:
        while tc < termination_condition:
            if eval_solution(solution) == 100:
                return solution

            n1 = update_forbidden_constraint3(solution)
            n2 = update_forbidden_constraint3_2(solution)
            n3 = update_forbidden_constraint2(solution)
            n4 = update_forbidden_constraint2_2(solution)
            n5 = update_length_work_blocks_constraint(solution)
            n6 = update_day_off_constraint(solution)
            n7 = update_demand_constraint(solution)

            neighborhood = [n1, n2, n3, n4, n5, n6, n7]

            neighbor = random.choice(neighborhood)

            if eval_solution(solution) < eval_solution(neighbor): solution = neigh
            elif random.uniform(0, 1) < exp_probability(solution, neighbor, T): sc

            tc += 1

        T *= r
        t += 1

    return "Not satified in the given time"
```

In [188]:

```
def test_contraints(solution):
    return {
            'demand_constraint': demand_constraint(solution),
            'day_off_constraint': day_off_constraint(solution),
            'length_work_blocks_constraint': length_work_blocks_constraint(solutic
            'forbidden_constraint2': forbidden_constraint2(solution),
            'forbidden_constraint3': forbidden_constraint3(solution)
            }
```

In [460]:

```
solution = simulated_annealing(input_data, 1000000, 10000, 0.99, 10000)
solution
```

```
[['D', 'D', 'D', 'D', 'D', 'D', '-'],
 ['A', 'A', 'A', '-', 'A', 'A', 'A'],
 ['D', 'D', '-', 'D', 'A', 'A', 'A'],
 ['D', 'D', 'D', 'A', 'N', 'N', '-'],
 ['-', 'A', 'N', 'N', 'N', 'N', 'N'],
 ['D', 'D', 'D', 'A', 'A', 'A', '-'],
 ['A', 'A', 'A', 'A', 'A', 'A', '-'],
 ['A', 'A', 'A', 'A', 'A', 'A', '-'],
 ['D', 'D', 'D', 'D', 'D', '-', 'N'],
 ['D', '-', 'D', 'A', 'A', 'A', '-'],
 ['-', 'D', 'D', 'D', 'A', 'A', 'N'],
 ['N', 'N', 'N', 'N', 'N', 'N', '-'],
 ['A', 'A', 'A', '-', 'A', 'A', 'A']]
```

In [95]:

```
pprint(test_contraints(solution))
```

```
{'day_off_constraint': True,
 'demand_constraint': True,
 'forbidden_constraint2': True,
 'forbidden_constraint3': True,
 'length_work_blocks_constraint': True}
```