In [1]:

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

import numpy as np
```

In [2]:

```
def read_data(filename):
    input_data = {}
    #Length of the schedule
    length_of_schedule = 0

    #Number of Employees
    number_of_employees = 0

    ##Number of Shifts
    number_of_shifts = 0

    # Temporal Requirements Matrix Shifts per Days
    temporal_requirements_matrix = []

    #ShiftName, Start, Length, MinlengthOfBlocks, MaxLengthOfBlocks
    shift_name, start_shift, length_shift, min_length_of_blocks, max_length_of_b
        = [], [], [], [], []

    # Minimum and maximum length of days-off blocks
    min_days_off = 0
    max_days_off = 0

    # Minimum and maximum length of work blocks
    min_length_work_blocks = 0
    max_length_work_blocks = 0

    # Number of not allowed shift sequences: NrSequencesOfLength2, NrSequencesOf
    nr_sequences_of_length_2 = 0
    nr_sequences_of_length_3 = 0

    # Not allowed shift sequences
    not_allowed_shif_sequences = [
        ['N', 'D'], ['N', 'A'], ['A', 'D']
    ]
    with open(filename, 'r') as f:
        lines = iter(f.readlines())
        for line in lines:
            if "#Length of the schedule" in line:
                length_of_schedule = int(next(lines))
            if "#Number of Employees" in line:
                number_of_employees = int(next(lines))
            if "##Number of Shifts" in line:
```

```
            number_of_shifts = int(next(lines))
        if "# Temporal Requirements Matrix" in line:
            ns = number_of_shifts
            temporal_requirements_matrix = []
            for i in range(number_of_shifts):
                temporal_requirements_matrix.append(list(map(int, next(lines
        if "#ShiftName" in line:
            ns = number_of_shifts
            shift_name, start_shift, length_shift, min_length_of_blocks, max
                = ['-']*ns, [0]*ns, [0]*ns, [0]*ns, [0]*ns
            for i in range(number_of_shifts):
                shift_name[i], start_shift[i], length_shift[i], min_length_o
                start_shift[i], length_shift[i] = list(map(int, start_shift[
            min_length_of_blocks, max_length_of_blocks = [int(x) for x in mi
        if "# Minimum and maximum length of days-off blocks" in line:
            min_days_off, max_days_off = list(map(int, next(lines).split()))
        if "# Minimum and maximum length of work blocks" in line:
            min_length_work_blocks, max_length_work_blocks = list(map(int, n
        if "# Number of not allowed shift sequences: NrSequencesOfLength2, N
            nr_sequences_of_length_2, nr_sequences_of_length_3 = list(map(in
        if "# Not allowed shift sequences" in line:
            not_allowed_shift_sequences_2, not_allowed_shift_sequences_3 = [
            for i in range(nr_sequences_of_length_2):
                not_allowed_shift_sequences_2.append(next(lines).split('\n'))
            for i in range(nr_sequences_of_length_3):
                not_allowed_shift_sequences_3.append(next(lines).split('\n'))

input_data = {
    'length_of_schedule': length_of_schedule,
    'number_of_employees': number_of_employees,
    'number_of_shifts': number_of_shifts,
    'temporal_requirements_matrix': temporal_requirements_matrix,
    'shift_name': shift_name,
    'start_shift': start_shift,
    'length_shift': length_shift,
    'min_length_of_blocks': min_length_of_blocks,
    'max_length_of_blocks': max_length_of_blocks,
    'min_days_off': min_days_off,
    'max_days_off': max_days_off,
    'min_length_work_blocks': min_length_work_blocks,
    'max_length_work_blocks': max_length_work_blocks,
    'nr_sequences_of_length_2': nr_sequences_of_length_2,
    'nr_sequences_of_length_3': nr_sequences_of_length_3,
```

```
        'not_allowed_shift_sequences_2': not_allowed_shift_sequences_2,
        'not_allowed_shift_sequences_3': not_allowed_shift_sequences_3
    }

    return input_data
```

In [3]:

```
def generate_random_solution(input_data):
    code = input_data['shift_name'] + ['-']
    return [[random.choice(code) for d in range(input_data['length_of_schedule']
```

In [4]:

```
nd_constraint(result, input_data):
e in range(input_data['number_of_employees']):
for d in range(input_data['length_of_schedule']):
    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 >= input_data['temporal_requirements_matrix'][0][d] and sum_afternoon :
    pass
else:
    return False
rn True
```

In [5]:

```
def update_demand_constraint(result, input_data):
    updated_result = deepcopy(result)
    for d in range(input_data['length_of_schedule']):
        sum_day, sum_afternoon, sum_night = 0, 0, 0
        for e in range(input_data['number_of_employees']):
            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 >= input_data['temporal_requirements_matrix'][0][d]: pass
        else: updated_result[random.randint(0, input_data['number_of_employees']
        if sum_afternoon >= input_data['temporal_requirements_matrix'][1][d]: pa
        else: updated_result[random.randint(0, input_data['number_of_employees']
        if sum_night >= input_data['temporal_requirements_matrix'][2][d]: pass
        else: updated_result[random.randint(0, input_data['number_of_employees']
    return updated_result
```

In [6]:

```
#day off constraint
def day_off_constraint(result, input_data):
    for e in range(input_data['number_of_employees']):
        count_dayoff = 0
        for d in range(input_data['length_of_schedule']):
            if result[e][d] == '-': count_dayoff += 1
        if input_data['min_days_off'] <= count_dayoff <= input_data['max_days_of
        else: return False
    return True
```

In [7]:

```
#day off constraint
def update_day_off_constraint(result, input_data):
    updated_result = deepcopy(result)
    code = input_data['shift_name'] + ['-']
    for e in range(input_data['number_of_employees']):
        count_dayoff = 0
        for d in range(input_data['length_of_schedule']):
            if updated_result[e][d] == '-': count_dayoff += 1
        if count_dayoff >= input_data['min_days_off']: pass
        else: updated_result[e][random.randint(0, input_data['length_of_schedule
        if count_dayoff <= input_data['max_days_off']: pass
        else: updated_result[e][random.randint(0, input_data['length_of_schedule
    return updated_result
```

In [8]:

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

In [9]:

```
ng days in a row constraint
date_length_work_blocks_constraint(result, input_data):
dated_result = deepcopy(result)
de = input_data['shift_name'] + ['-']
r e in range(input_data['number_of_employees']):
  count_consecutive = 0
  min_flag, max_flag = False, False
  for d in range(input_data['length_of_schedule'] - 1):
      if updated_result[e][d] != '-' and updated_result[e][d+1] != '-':
          count_consecutive += 1
          if count_consecutive >= input_data['min_length_work_blocks'] - 1:
              min_flag = True
              if count_consecutive <= input_data['max_length_work_blocks'] - 1:
                  max_flag = True
      else:  count_consecutive = 0

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

turn updated_result
```

In [10]:

```
#forbidden shifts constraint
def forbidden_constraint2(result, input_data):
    if input_data['not_allowed_shift_sequences_2'] == []: return True
    if input_data['not_allowed_shift_sequences_2'] != []:
        for e in range(input_data['number_of_employees']):
            for d in range(input_data['length_of_schedule'] - 1):
                for f in input_data['not_allowed_shift_sequences_2']:
                    if result[e][d] == f[0] and result[e][d+1] == f[1]: return F
        return True
```

In [11]:

```
#forbidden shifts constraint
def update_forbidden_constraint2(result, input_data):
    updated_result = deepcopy(result)
    code = input_data['shift_name'] + ['-']
    if input_data['not_allowed_shift_sequences_2'] == []: return updated_result
    if input_data['not_allowed_shift_sequences_2'] != []:
        for e in range(input_data['number_of_employees']):
            for d in range(input_data['length_of_schedule'] - 1):
                for f in input_data['not_allowed_shift_sequences_2']:
                    if updated_result[e][d] == f[0] and updated_result[e][d+1] =
                        for c in code:
                            if [f[0], c] not in input_data['not_allowed_shift_se
                                updated_result[e][d+1] = c
                                break
        return updated_result
```

In [12]:

```
#forbidden shifts constraint
def update_forbidden_constraint2_2(result, input_data):
    updated_result = deepcopy(result)
    code = input_data['shift_name'] + ['-']
    if input_data['not_allowed_shift_sequences_2'] == []: return updated_result
    if input_data['not_allowed_shift_sequences_2'] != []:
        for e in range(input_data['number_of_employees']):
            for d in range(input_data['length_of_schedule'] - 1):
                for f in input_data['not_allowed_shift_sequences_2']:
                    if updated_result[e][d] == f[0] and updated_result[e][d+1] =
                        for c in code:
                            if [c, f[1]] not in input_data['not_allowed_shift_se
                                updated_result[e][d] = c
                                break
        return updated_result
```

In [13]:

```
def forbidden_constraint3(result, input_data):
    if input_data['not_allowed_shift_sequences_3'] == []: return True
    if input_data['not_allowed_shift_sequences_3'] != []:
        for e in range(input_data['number_of_employees']):
            for d in range(input_data['length_of_schedule'] - 2):
                for f in input_data['not_allowed_shift_sequences_3']:
                    if result[e][d] == f[0] and result[e][d+1] == f[1] \
                                            and result[e][d+2] == f[2]: return F
        return True
```

In [14]:

```
def update_forbidden_constraint3(result, input_data):
    updated_result = deepcopy(result)
    code = input_data['shift_name'] + ['-']
    if input_data['not_allowed_shift_sequences_3'] == []: return updated_result
    if input_data['not_allowed_shift_sequences_3'] != []:
        for e in range(input_data['number_of_employees']):
            for d in range(input_data['length_of_schedule'] - 2):
                for f in input_data['not_allowed_shift_sequences_3']:
                    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 input_data['not_allowed_sh
                                updated_result[e][d] = c
                                break

        return updated_result
```

In [15]:

```
def update_forbidden_constraint3_2(result, input_data):
    updated_result = deepcopy(result)
    code = input_data['shift_name'] + ['-']
    if input_data['not_allowed_shift_sequences_3'] == []: return updated_result
    if input_data['not_allowed_shift_sequences_3'] != []:
        for e in range(input_data['number_of_employees']):
            for d in range(input_data['length_of_schedule'] - 2):
                for f in input_data['not_allowed_shift_sequences_3']:
                    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 input_data['not_allowed_sh
                                updated_result[e][d] = c
                                break

        return updated_result
```

In [16]:

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

    return score
```

In [17]:

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

In [18]:

```
simulated_annealing(input_data, T_max, r, termination_condition, halting_conditi
tc = 0
hc = 0
score = 100
T = T_max

solution = generate_random_solution(input_data)

while hc < halting_condition:
    while tc < termination_condition:
        if eval_solution(solution, input_data) == 100:
            return solution

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

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

        neighbor = random.choice(neighborhood)

        if eval_solution(solution, input_data) < eval_solution(neighbor, input_d
        elif random.uniform(0, 1) < exp_probability(solution, neighbor, T, input

        tc += 1

    T *= r
    hc += 1

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

In [31]:

```
def eval_SA_params(input_data, T_max, tc_max, hc_max, time_limit):
    print(f"Running SA on {input_data['filename']}")
    params = []
    start_time_limit = time.time()
    for T in range(1, T_max):
        end_time_limit = time.time()
        if end_time_limit - start_time_limit > time_limit:
            break
        for hc in range(1, hc_max):
            end_time_limit = time.time()
            if end_time_limit - start_time_limit > time_limit:
                break
            for tc in range(1, tc_max):
                end_time_limit = time.time()
                if end_time_limit - start_time_limit > time_limit:
                    print(f'Exceeded time limit of {time_limit} seconds.')
                    break
                start = time.time()
                solution = simulated_annealing(input_data, T, 0.99, tc, hc)
                end = time.time()
                if solution != "Not satified in the given time":
                    print(f'Parameters solution T={T}\tr={0.99}\ttermination_con
                    params.append((T, r/100, tc, hc, end - start))
    return params
```

In [20]:

```
def get_examples():
    RWS_INSTANCES = 'rws_instances/'
    EXAMPLE = 'Example'
    filenames = []

    for i in range(1,21):
        filenames.append(RWS_INSTANCES + EXAMPLE + str(i) + '.txt')
    return filenames
```

In [ ]:

```
def run_SA_eval_on_examples():
    T = 100
    tc = 100
    hc = 100
    time_limit = 100
    params = {}
    for example in get_examples():
        try:
            input_data = read_data(example)
            input_data['filename'] = example
            params[example] = eval_SA_params(input_data, T, tc, hc, time_limit)
        except Exception as e:
            print(e)
    return params
```

In [22]:

```
T = 100
r = 0.99
tc = 1000
hc = 1000
solution = simulated_annealing(read_data(get_examples()[3]), T, r/100, tc, hc)
solution
```

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

In [44]:

```
def run_SA_on_examples():
    T = 100
    r = 0.99
    tc = 1000
    hc = 1000
    solutions = {}
    for example in get_examples():
        try:
            solution = simulated_annealing(read_data(example), T, r/100, tc, hc)
            print(example)
            pprint(solution)
            solutions[example] = solution
        except Exception as e:
            print(e)
```

In [45]:

```
solutions = run_SA_on_examples()
```

```
rws_instances/Example1.txt
'Not satified in the given time'
rws_instances/Example2.txt
'Not satified in the given time'
rws_instances/Example3.txt
'Not satified in the given time'
rws_instances/Example4.txt
[['-', '-', 'D', '-', 'A', 'A', 'A'],
 ['D', 'D', '-', 'D', 'A', 'N', 'N'],
 ['-', 'D', 'A', 'A', 'A', 'A', '-'],
 ['-', 'D', 'D', 'D', 'A', 'N', '-'],
 ['D', 'D', 'D', 'D', 'D', 'D', '-'],
 ['D', 'D', '-', 'D', 'D', 'N', 'N'],
 ['D', 'A', 'A', 'A', '-', 'A', 'N'],
 ['D', 'D', '-', 'D', 'D', 'A', 'N'],
 ['D', 'D', 'D', 'D', '-', 'N', 'N'],
 ['D', 'D', 'D', '-', 'D', 'N', 'N'],
 ['D', '-', 'D', 'D', 'D', 'D', 'N'],
 ['A', 'A', 'A', 'A', '-', 'A', 'N'],
 ['D', 'D', 'D', 'D', '-', 'D', 'N']]
rws_instances/Example5.txt
'Not satified in the given time'
rws_instances/Example6.txt
'Not satified in the given time'
rws_instances/Example7.txt
'Not satified in the given time'
rws_instances/Example8.txt
'Not satified in the given time'
list index out of range
rws_instances/Example10.txt
'Not satified in the given time'
rws_instances/Example11.txt
'Not satified in the given time'
list index out of range
rws_instances/Example13.txt
'Not satified in the given time'
rws_instances/Example14.txt
'Not satified in the given time'
rws_instances/Example15.txt
'Not satified in the given time'
rws_instances/Example16.txt
'Not satified in the given time'
list index out of range
rws_instances/Example18.txt
'Not satified in the given time'
rws_instances/Example19.txt
'Not satified in the given time'
rws_instances/Example20.txt
'Not satified in the given time'
```