```
from random import randint
from random import random
from random import shuffle

# number of jobs
N = 10
# number of machines
M = 3
# Other (M, N) combinations:
# (3, 10), (4, 15), (5, 24), (6, 32), (7, 26), (8, 42), (9, 56)

# max number of modes
MAX_MODES = 5
# total number of resources
R = M + 1

ALPHA = 0.6

jobs = list(range(1, N + 1))
jobs
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
modes = [randint(1, MAX_MODES) for job in jobs ]
modes
```

```
[3, 1, 2, 2, 1, 2, 3, 4, 5, 4]
```

```
proc_time = []
for job in jobs:
  initial_time = randint(10, 50) # for first mode
  proc_time.append(
      [
       round(initial_time * (1 -(ALPHA * (1 - 1 / k) )), 2)
        for k in range(1, modes[job - 1] + 1)
      ]
  )
proc_time
```

```
[[17.0, 11.9, 10.2],
 [24.0],
 [16.0, 11.2],
 [50.0, 35.0],
 [36.0],
 [45.0, 31.5],
 [22.0, 15.4, 13.2],
 [42.0, 29.4, 25.2, 23.1],
 [24.0, 16.8, 14.4, 13.2, 12.48],
 [40.0, 28.0, 24.0, 22.0]]
```

```
r = []
a = 5
b = 1
# proc time is a function of allocated resources
```

```
# proc_time is a function of allocated resources
for i in range(len(proc_time)):
  r.append([round(1 / proc_time[i][j] * a + b, 2) for j in range(len(proc_time[i]))
r
```

```
[[1.29, 1.42, 1.49],
 [1.21],
 [1.31, 1.45],
 [1.1, 1.14],
 [1.14],
 [1.11, 1.16],
 [1.23, 1.32, 1.38],
 [1.12, 1.17, 1.2, 1.22],
 [1.21, 1.3, 1.35, 1.38, 1.4],
 [1.12, 1.18, 1.21, 1.23]]
```

```
def init_master():
  return [randint(1, modes[job - 1]) for job in jobs]

init_master()
```

```
[2, 1, 2, 1, 1, 1, 1, 3, 2, 3]
```

```
def init_slave_random():
  perm = [str(job) for job in jobs]
  for i in range(M - 1):
    perm.append('*')
  shuffle(perm)

  # prevent idle machines
  if perm[0] == '*' or perm[-1] == '*':
    return init_slave_random()
  for i in range(1, len(perm)):
    if perm[i - 1] == '*' and perm[i] == '*':
      return init_slave_random()
  return perm

init_slave_random()
```

```
['3', '1', '8', '7', '10', '5', '9', '*', '6', '*', '4', '2']
```

```
import numpy as np

def init_slave(master):
  # sort by longest proc_time first
  p_times = [-1 * proc_time[i][master[i] - 1] for i in range(len(master))]
  arg_sort_jobs = np.argsort(p_times)
  chr = []
  for i in range(len(arg_sort_jobs)):
    chr.append(str(jobs[arg_sort_jobs[i]]))
    if i % (M + 2) == 0:
      chr.append('*')
  return chr

init_slave(init_master())
```

```
['4', '*', '5', '6', '8', '10', '2', '*', '9', '7', '3', '1']
```

```
from random import sample
def lambda_interchange(slave):
  i, j  = sample(range(len(slave)), 2)
  i, j = (j, i) if i > j else (i, j)

  # try again
  if slave[i] == '*' or slave[j] == '*'  or j == len(slave) - 1:
    return lambda_interchange(slave)
  i_end = i + 1
  j_end = j + 1

  for k in range(i, len(slave) - 1):
    if slave[k + 1] == '*' or k == j-1:
      break
    i_end += 1

  for k in range(j, len(slave) - 1):
    if slave[k] == '*':
      break
    j_end += 1

  return slave[:i] + slave[j:j_end] + slave[i_end : j] + slave[i:i_end] + slave[j_e
sl = init_slave(init_master())
print(sl)
print(lambda_interchange(sl))
```

```
['4', '*', '6', '10', '5', '8', '2', '*', '9', '3', '7', '1']
['4', '*', '6', '10', '2', '*', '5', '8', '9', '3', '7', '1']
```

```
NP = 4 # population size
NS = 4 # number of slaves per master
def init_population():
  master = init_master()
  slave = init_slave(master)
  population = [lambda_interchange(slave) for _ in range(NS)]
  return (slave, population)

population = [init_population() for _ in range(NP)]
population
```

```
     [(['1', '*', '4', '10', '6', '7', '2', '*', '8', '3', '5', '9'],
       ['6', '7', '2', '*', '*', '4', '10', '1', '8', '3', '5', '9'],
       ['1', '*', '3', '5', '9', '*', '8', '4', '10', '6', '7', '2'],
       ['4', '10', '6', '7', '2', '*', '*', '1', '8', '3', '5', '9'],
       ['1', '*', '10', '6', '7', '2', '*', '4', '8', '3', '5', '9']]),
      (['1', '*', '4', '6', '7', '10', '2', '*', '8', '5', '3', '9'],
       ['1', '*', '4', '10', '2', '*', '6', '7', '8', '5', '3', '9'],
       ['1', '*', '6', '7', '10', '2', '*', '4', '8', '5', '3', '9'],
       ['1', '*', '8', '5', '3', '9', '*', '4', '6', '7', '10', '2'],
       ['1', '*', '4', '6', '7', '10', '5', '3', '9', '*', '8', '2']]),
      (['1', '*', '4', '2', '6', '7', '10', '*', '8', '9', '3', '5'],
       ['1', '*', '10', '*', '4', '2', '6', '7', '8', '9', '3', '5'],
       ['1', '*', '8', '9', '3', '5', '*', '4', '2', '6', '7', '10'],
       ['1', '*', '4', '9', '3', '5', '*', '8', '2', '6', '7', '10'],
       ['1', '*', '4', '2', '8', '9', '3', '5', '*', '6', '7', '10']]),
      (['1', '*', '4', '7', '2', '6', '10', '*', '8', '3', '5', '9'],
       ['1', '*', '4', '7', '8', '3', '5', '9', '*', '2', '6', '10'],
       ['1', '*', '4', '2', '6', '10', '*', '7', '8', '3', '5', '9'],
       ['1', '*', '4', '7', '5', '9', '*', '8', '3', '2', '6', '10'],
       ['6', '10', '*', '*', '4', '7', '2', '1', '8', '3', '5', '9']])]
```

```
MUTATION_RATE = 0.95
def mutate_master(master):
  # select random job and assign random mutation
  if random() >= MUTATION_RATE:
    return master
  mutate_idx = randint(0, len(master) - 1)
  master[mutate_idx] = randint(1, modes[mutate_idx])
  return master

master = init_master()
print(master)
mutate_master(master)
print(master)
```

```
[3, 1, 2, 2, 1, 1, 3, 3, 4, 1]
[3, 1, 2, 2, 1, 1, 3, 3, 4, 4]
```

```
def mutate_slave(slave):
  # perform swap mutation
  if random() >= MUTATION_RATE:
    return master
  i, j  = sample(range(len(slave)), 2)
  slave[i], slave[j] = slave[j], slave[i]
  for i in range(1, len(slave)):
    if slave[0] == '*' or slave[-1] == '*' or (slave[i - 1] == slave[i] and slave[i
      mutate_slave()
      break
slave = init_slave_random()
print(slave)
mutate_slave(slave)
print(slave)
```

```
['8', '5', '6', '3', '10', '1', '4', '9', '*', '2', '*', '7']
['8', '2', '6', '3', '10', '1', '4', '9', '*', '5', '*', '7']
```

```
def cross_over_slaves(slave_1, slave_2):
  # apply partial cross over mapping
  i, j  = sample(range(len(slave_1)), 2)
  i, j = (j, i) if i > j else (i, j)

  for k in range(i, j + 1):
    if slave_1[k] == '*' or slave_2[k] == '*':
      # try again
      cross_over_slaves(slave_1, slave_2)
      return

  p_map = {}

  for k in range(i, j + 1):
    p_map[slave_1[k]] = slave_2[k]
    p_map[slave_2[k]] = slave_1[k]

  for k in range(len(slave_1)):
    if slave_1[k] in p_map:
      slave_1[k] = p_map[slave_1[k]]

    if slave_2[k] in p_map:
      slave_2[k] = p_map[slave_2[k]]

slave_1 = init_slave_random()
slave_2 = init_slave_random()
print(slave_1)
print(slave_2)
cross_over_slaves(slave_1, slave_2)
print("After cross over")
print(slave_1)
print(slave_2)
```

```
['7', '*', '2', '5', '10', '4', '9', '*', '8', '3', '6', '1']
['9', '1', '8', '10', '*', '2', '7', '*', '5', '4', '6', '3']
After cross over
['7', '*', '2', '5', '10', '3', '9', '*', '8', '1', '6', '3']
['9', '3', '8', '10', '*', '2', '7', '*', '5', '3', '6', '1']
```

```
def calculate_fitness(master, slave):
  NaN = float('nan')
  t = 0
  # start indexing by 1
  # p[i] - processing time for job i
  p = [NaN] + [proc_time[i][master[i] - 1] for i in range(len(master))]
  # r[i] - resources taken for job i

  res = [NaN] + [r[i][master[i] - 1] for i in range(len(master))]

  # set of scheduled jobs on machine i
  scheduled = [[NaN]] + [[NaN] for i in range(M)]
  # number of unscheduled jobs on machine i
  unscheduled = [[NaN]] + [[NaN] for i in range(M)]
```

```
# index o of the current assign job on machine j
sigma = [NaN] + [1 for j in range(M)]

# pi[j] = i -- job i assigned on machine j
pi = [[NaN]] + [[NaN] for i in range(M)]
machine_idx = 1
for job in slave:
  if job == '*':
    machine_idx += 1
  else:
    unscheduled[machine_idx].append(int(job))
    pi[machine_idx].append(int(job))

# the available time on machine i
c = [NaN] + [0 for i in range(M)]
# starting times
tau = [NaN] + [NaN for i in range(N)]

print(pi)
print(unscheduled)


Rt = R - sum([res[scheduled[j]] \
              for j in range(1, M + 1) \
              for i in range(1, len(scheduled[j]))])

while True:
  # obtain the available machines at time t
  Mt = [NaN]
  for j in range(1, M + 1):
    if len(unscheduled[j]) == 1: # default is one Nan elem
      continue
    if res[pi[j][sigma[j]]] <= Rt and c[j] <= t:
      Mt.append(j)


  while len(Mt) > 1:
    if sum([res[pi[j][sigma[j]]] for j in Mt if j == j]) <= Rt:
      # every available machine is aranged
      for j in range(1, len(Mt)):
        tau[pi[j][sigma[j]]] = t
        c[j] = t + p[pi[j][sigma[j]]]
        scheduled[j].append(pi[j][sigma[j]])
        unscheduled[j].remove(pi[j][sigma[j]])
        sigma[j] += 1
    else:
        # calculate left workload
        P = [NaN] + [sum([i for i in unscheduled[j] if i == i]) for j in Mt if j

        # calculate priority
        w = [0] + [P[j] + p[pi[j][sigma[j]]] / res[pi[j][sigma[j]]] for j in Mt i

        j_prime = np.argmax(w)
        j_prime = Mt[j_prime]
        tau[pi[j_prime][sigma[j_prime]]] = t
        c[j_prime] = t + p[pi[j_prime][sigma[j_prime]]]
```

```
            c[j_prime] = t + p[pi[j_prime][sigma[j_prime]]]
            scheduled[j_prime].append(pi[j_prime][sigma[j_prime]])
            unscheduled[j_prime].remove(pi[j_prime][sigma[j_prime]])
            Rt -= res[pi[j_prime][sigma[j_prime]]]
            to_remove = [j_prime] + [Mt[j] for j in range(1, len(Mt)) if res[pi[j_pri

            for elem in to_remove:
              Mt.remove(elem)
            sigma[j_prime] += 1

    if scheduled == pi:
      return max([c[j] for j in range(1, M + 1)])

master = init_master()
slave = init_slave(master)
print(calculate_fitness(master, slave))


def run_algorithm():
  MAX_ITER = 100
  Best = None
  population = init_population()
  CRs = len(population) / 2
  for _ in range(MAX_ITER):
    fitness = [calculate_fitness(master, slave) for (master, slave) in population]
    for (master, slave) in population:
        mutate_master(master)
        mutate_slave(slave)
    for _ in range(CRs):
        (m1, s1), (m2, s2) = sample(population, 2)
        cross_over_slaves(s1, s2)
    Best = max(fitnesses)
```