**Cuckoo Algorithm**

**Job Scheduling:**

import numpy as np

import random

class CuckooSearch:

    def \_\_init\_\_(self, jobs, machines, n\_nests, n\_iterations, alpha=0.01, beta=1.5):

        self.jobs = jobs

        self.machines = machines

        self.n\_nests = n\_nests

        self.n\_iterations = n\_iterations

        self.alpha = alpha

        self.beta = beta

        self.n\_jobs = len(jobs)

        self.best\_solution = None

        self.best\_makespan = float('inf')

    def run(self):

        nests = [self.random\_solution() for \_ in range(self.n\_nests)]

        for \_ in range(self.n\_iterations):

            for nest in nests:

                new\_nest = self.cuckoo(nest)

                new\_makespan = self.makespan(new\_nest)

                if new\_makespan < self.makespan(nest):

                    nest[:] = new\_nest

                if new\_makespan < self.best\_makespan:

                    self.best\_makespan = new\_makespan

                    self.best\_solution = new\_nest

        return self.best\_solution, self.best\_makespan

    def cuckoo(self, nest):

        new\_nest = nest[:]

        for i in range(len(nest)):

            if random.random() < 0.5:

                new\_nest[i] = int(nest[i] + self.alpha \* (random.random() - 0.5))

            else:

                new\_nest[i] = int(nest[i] + self.beta \* (random.random() - 0.5))

            # Ensure the machine index is within bounds

            new\_nest[i] = max(0, min(new\_nest[i], self.machines - 1))

        return new\_nest

    def random\_solution(self):

        solution = []

        for j in range(self.n\_jobs):

            solution.append(random.randint(0, self.machines - 1))

        return solution

    def makespan(self, solution):

        machine\_times = [0] \* self.machines

        job\_times = [0] \* self.n\_jobs

        for job, machine in enumerate(solution):

            start\_time = max(machine\_times[machine], job\_times[job])

            job\_times[job] = start\_time + self.jobs[job][machine]

            machine\_times[machine] = job\_times[job]

        return max(job\_times)

jobs = [

    [3, 2, 2],

    [2, 3, 2],

    [4, 3, 3]

]

machines = 3

n\_nests = 10

n\_iterations = 100

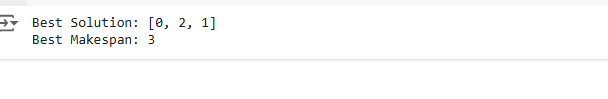
csa = CuckooSearch(jobs, machines, n\_nests, n\_iterations)

best\_solution, best\_makespan = csa.run()

print("Best Solution:", best\_solution)

print("Best Makespan:", best\_makespan)

**OUTPUT:**

****