```
import numpy as np
import random
import matplotlib.pyplot as plt

# Define the Job-Shop Scheduling Problem
# Processing time for each job on each machine
processing_time = [
    [3, 2, 2],  # Job 1
    [2, 1, 4],  # Job 2
    [4, 3, 2]   # Job 3
]

n_jobs = len(processing_time)  # Number of jobs
n_machines = len(processing_time[0])  # Number of machines

# Fitness function for Job-Shop Scheduling (Makespan)
def fitness_schedule(schedule):
    # Create a list to track when each machine is available
    machine_times = [0] * n_machines  # Start times for each machine
    job_end_times = [0] * n_jobs  # End times for each job

    # Schedule jobs
    for job_order in schedule:
        for machine_id, operation in enumerate(job_order):
            operation = int(operation)  # Ensure operation is an integer
            start_time = max(machine_times[machine_id], job_end_times[operation])  # When the machine and job are ready
            end_time = start_time + processing_time[operation][machine_id]
            machine_times[machine_id] = end_time  # Update machine's availability
            job_end_times[operation] = end_time  # Update job's end time

    # Makespan is the time when the last job finishes
    makespan = max(machine_times)
    return makespan

# Grey Wolf Optimization (GWO) for Job-Shop Scheduling
def gwo_job_shop(fitness, max_iter, n, bounds):
    # Initializing population with random schedules
    population = []
    for _ in range(n):
        # Randomly create a schedule (a list of job orders on machines)
        schedule = [list(np.random.permutation(n_jobs)) for _ in range(n_machines)]  # Random job order for each machine
        population.append(schedule)

    # Sort the population based on fitness (makespan)
    population = sorted(population, key=lambda schedule: fitness(schedule))
    alpha, beta, gamma = population[:3]  # Best 3 solutions

    # Track the makespan for each iteration
    makespan_history = []

    for Iter in range(max_iter):
        a = 2 * (1 - Iter / max_iter)  # Decreasing exploration rate

        for i in range(n):
            A1, A2, A3 = a * (2 * random.random() - 1), a * (2 * random.random() - 1), a * (2 * random.random() - 1)
            C1, C2, C3 = 2 * random.random(), 2 * random.random(), 2 * random.random()

            # Update the wolf's position (schedule) based on the alpha, beta, gamma wolves
            X1 = np.array(alpha) - A1 * np.abs(C1 * np.array(alpha) - np.array(population[i]))
            X2 = np.array(beta) - A2 * np.abs(C2 * np.array(beta) - np.array(population[i]))
            X3 = np.array(gamma) - A3 * np.abs(C3 * np.array(gamma) - np.array(population[i]))
            Xnew = (X1 + X2 + X3) / 3

            # Ensure that the updated schedule contains integer indices and stays within valid bounds
            Xnew = np.round(Xnew).astype(int)  # Round and convert to integers
            Xnew = np.clip(Xnew, 0, n_jobs - 1)  # Ensure indices are within valid bounds

            # Ensure the new solution is a valid schedule (a list of job orders for each machine)
            for machine_id in range(n_machines):
                Xnew[machine_id] = list(np.random.permutation(n_jobs))

            # Evaluate new solution
            fnew = fitness(Xnew)
            if fnew < fitness(population[i]):
                population[i] = Xnew
```

```
        # Sort and update the best solutions
        population = sorted(population, key=lambda schedule: fitness(schedule))
        alpha, beta, gamma = population[:3]  # Update alpha, beta, gamma wolves

        # Track the makespan at each iteration
        makespan_history.append(fitness(alpha))  # Use the fitness of the alpha wolf (best solution)

    return alpha, makespan_history  # Return the best schedule and makespan history

# Example usage for Job-Shop Scheduling Problem
bounds = [(0, n_jobs-1)] * n_machines  # Bounds for each machine (jobs can be scheduled in any order)
n = 50  # Number of wolves in the population
max_iter = 100  # Number of iterations

best_schedule, makespan_history = gwo_job_shop(fitness_schedule, max_iter, n, bounds)

# Output the best schedule
print("Best schedule (jobs order on each machine):")
for machine_id in range(n_machines):
    print(f"Machine {machine_id + 1}: {best_schedule[machine_id]}")

# Fitness of the best solution (makespan)
best_makespan = fitness_schedule(best_schedule)
print(f"Best makespan (time to complete all jobs): {best_makespan}")

# Plot the makespan history over iterations
plt.plot(range(max_iter), makespan_history, label='Makespan')
plt.xlabel('Iteration')
plt.ylabel('Makespan (Time to complete all jobs)')
plt.title('Makespan Evolution during GWO Optimization')
plt.grid(True)
plt.legend()
plt.show()
```

```
Best schedule (jobs order on each machine):
Machine 1: [1 0 2]
Machine 2: [1 0 2]
Machine 3: [1 0 2]
Best makespan (time to complete all jobs): 6
```